المملكـة العربيــة الســعودية
المؤسسة العامة للتعليم الفني والتدريب المهني
الإدارة العامة لتصميم وتطوير المناهج



دعم فني

برمجة الحاسب

١٤١ حاب



الحمد لله وحده، والصلاة والسلام على من لا نبي بعده، محمد وعلى آله وصحبه، وبعد:

تسعى المؤسسة العامة للتعليم الفني والتدريب المهني لتأهيل الكوادر الوطنية المدربة القادرة على شغل الوظائف التقنية والفنية والمهنية المتوفرة في سوق العمل، ويأتي هذا الاهتمام نتيجة للتوجهات السديدة من لدن قادة هذا الوطن التي تصب في مجملها نحو إيجاد وطن متكامل يعتمد ذاتياً على موارده وعلى قوة شبابه المسلح بالعلم والإيمان من أجل الاستمرار قدماً في دفع عجلة التقدم التنموي: لتصل بعون الله تعالى لمصاف الدول المتقدمة صناعياً.

وقد خطت الإدارة العامة لتصميم وتطوير المناهج خطوة إيجابية تتفق مع التجارب الدولية المتقدمة في بناء البرامج التدريبية، وفق أساليب علمية حديثة تحاكي متطلبات سوق العمل بكافة تخصصاته لتلبي متطلباته ، وقد تمثلت هذه الخطوة في مشروع إعداد المعايير المهنية الوطنية الذي يمثل الركيزة الأساسية في بناء البرامج التدريبية، إذ تعتمد المعايير في بنائها على تشكيل لجان تخصصية تمثل سوق العمل والمؤسسة العامة للتعليم الفني والتدريب المهني بحيث تتوافق الرؤية العلمية مع الواقع العملي الذي تفرضه متطلبات سوق العمل، لتخرج هذه اللجان في النهاية بنظرة متكاملة لبرنامج تدريبي أكثر التصاقاً بسوق العمل، وأكثر واقعية في تحقيق متطلباته الأساسية.

وتتناول هذه الحقيبة التدريبية " برمجة الحاسب " لمتدربي قسم" دعم فني " للكليات التقنية موضوعات حيوية تتناول كيفية اكتساب المهارات اللازمة لهذا التخصص.

والإدارة العامة لتصميم وتطوير المناهج وهي تضع بين يديك هذه الحقيبة التدريبية تأمل من الله عز وجل أن تسهم بشكل مباشر في تأصيل المهارات الضرورية اللازمة، بأسلوب مبسط يخلو من التعقيد، وبالاستعانة بالتطبيقات والأشكال التي تدعم عملية اكتساب هذه المهارات.

والله نسأل أن يوفق القائمين على إعدادها والمستفيدين منها لما يحبه ويرضاه: إنه سميع مجيب الدعاء.

الإدارة العامة لتصميم وتطوير المناهج



برمجة الحاسب

مقدمة و حل المشكلة

مقدمة و حل المشكلة

١

تمهيد

من المعلوم اليوم أن الحاسبات انتشرت انتشارا واسعا وكبيرا لدرجة أنها أصبحت في كل موقع وفي كل مكان ولا يمكن الاستغناء عنها بأي حال من الأحوال، وذلك لما تقوم به من أعمال كبيرة وعظيمة و لما تتمتع به من قدرة عالية على إجراء العمليات الحسابية وغيرها من العمليات في وقت قصير جدا، كما أنها تتميز بالقدرات العالية على معالجة الكم الهائل من البيانات حفظا وترتيبا واسترجاعا وبحثا وغيرها الكثير من العمليات.

ونظرا لما سبق أصبح لزاما علينا - لكي نواكب هذا العصر ولكي ننهض بوطننا وشعبنا و أمتنا - أن نعرف الكثير عن هذه الحاسبات وكيف يمكن التعامل معها والاستفادة منها. ومن الوسائل التي تساعدنا على الاستفادة من هذه الحاسبات معرفة وإتقان إحدى لغات البرمجة المعروفة والمشهورة هذه الأيام، ومن هذه اللغات المشهورة والتي بدأت تستخدم على نطاق واسع لغة الجافا Java language وذلك لما تتمتع به من قدرة على العمل (التنفيذ) مع كل الحاسبات وسهولة كتابة البرامج المختلفة سواء منها البسيطة أو الكبيرة.

وهذه الحقيبة تقدم شرحا تفصيليا للمفردات الأساسية المكونة للغة الجافا وكذلك كتابة بعض البرامج البسيطة والمتوسطة باستخدام هذه اللغة. ففي الوحدة الأولى مقدمة للغات البرمجة المختلفة وشرح لكيفية تحليل وحل المشاكل البسيطة باستخدام خرائط التدفق والكود الزائف وكذلك كتابة البرامج البسيطة ومعرفة المفردات الأساسية للغة من متغيرات وأنواع البيانات والعمليات الحسابية والمنطقية وغيرها من العمليات تم شرحها وتوضيحها في الوحدة الثانية. أما الوحدة الثالثة فإنها تتناول الحلقات (looping) بأنواعها المختلفة والتفريعات (branching) وكيفية كتابتها والاستفادة منها في حل البرامج البسيطة والمتوسطة، بالإضافة إلى تنفيذ هذه البرامج على الحاسب.

الوحدة الأولى

مقدمة وحل المشكلة

في هذه الوحدة نعرض مقدمة عن ماهية برنامج الحاسب ولغة البرمجة وأنواع لغات البرمجة وأهمية مهنة البرمجة، ثم بعد ذلك نشرح القواعد التي تساعد في تحليل المشكلة ومعرفة عناصرها المكونة لها و كيف يمكن تجزئة المشكلة إلى أجزاء صغيرة يسهل التعامل معها، وفيها أيضا نوضح رموز رسم خرائط التدفق ثم رسم هذه الخرائط للمشكلة بعد كتابة الخوارزم والتي تعطي صورة لحل المشكلة.

الفصل الأول: مقدمة

الجدارة:

معرفة ماهية برنامج الحاسب ولغات البرمجة وأنواعها

الأهداف:

عندما تكمل هذه الوحدة يكون لديك القدرة على:

١ - فهم ماهية برنامج الحاسب
٢ - معرفة لغات البرمجة المختلفة التي يمكن أن يراها
٣ - الإخبار بأهمية مهنة البرمجة
٤ - معرفة ما هو علم صناعة البرمجيات

مستوى الأداء المطلوب

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪

الوقت المتوقع للتدريب: ساعة واحدة

الوسائل المساعدة:

- قلم
- دفتر

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة

الفصل الأول: مقدمة

نظراً للتطور الكبير في تقنية صناعات الحاسبات الآلية وانتشارها في جميع مجالات الحياة المختلفة، واستخداماتها المتعددة في شتى المجالات، فإنه أصبح لزاماً علينا معرفة هذه الحاسبات وكيفية التعامل معها والاستفادة منها لأنها توفر الجهد والوقت وتنجز كثير من الأعمال بدقه كبيرة بالإضافة إلى قدراتها الكبيرة في الاحتفاظ بالبيانات. ومن الطرق الشائعة للاستفادة من القدرات الكبيرة للحاسبات هو: بناء البرامج التي تقوم بحل كثير من المشكلات توفيراً للجهد والوقت ووصولاً إلى الدقة المطلوبة، وفي هذه الوحدة سوف نلقي الضوء على ماهية برنامج الحاسب وكذلك أنواع لغات البرمجة المختلفة. ثم بعد ذلك نبين أهمية مهنة البرمجة وصناعة البرمجيات.

برنامج الحاسب

البرنامج هو عبارة عن مجموعة من التعليمات تعطى للحاسب للقيام بعمل ما مثل حساب مجموع قيم مختلفة، حساب المتوسط الحسابي، حساب مضروب عدد معين ....الخ

والبرنامج هو الذي يحدد للحاسب كيفية التعامل مع البيانات للحصول على النتائج المطلوبة. والبرنامج يكتب بواسطة المبرمج (Computer Programmer) الذي يفهم المشكلة ويقترح الحل وينفذه لحل هذه المشكلة ويجب أن يكون البرنامج في مجموعه صحيحاً وواضحا وليس فيه لبس أو غموض.

والبرمجيات (Software) هي التي تسهل للمستخدم استخدام المكونات المادية (Hardware) بكفاءة وراحة ويمكن تقسيم البرمجيات إلى ثلاثة أنواع رئيسية وهي: -

١ - برامج التشغيل Operating System

مثل النوافذ(windows) و Dos، Unix، Linux، VMS وغيرها. وهي عبارة عن برامج تقوم بدور الوسيط بين المستخدم والمكونات المادية وهي تمكن المستخدم من استخدام المكونات المادية للحاسب بكفاءة وبراحة، كما أنها تساعد المستخدم في إنشاء نظام الملفات وغيرها. ومن برامج التشغيل ما يصلح للعمل في الشبكات مثل Unix، Windows، ومنها الذي يستخدم مع الحاسب فقط مثل Dos.

٢ - **برامج التطبيقات** Application Programs

وهي برامج تساعد في إنشاء كثير من التطبيقات مثل إنشاء قاعدة بيانات والرسم باستخدام الحاسب و غيرها ومن أمثلة هذه البرامج: -

برنامج الأوتوكاد Autocad– الاكسيل Excel - الأكسس Access - الأوراكلOracle - الفوتوشوب Fotoshop وغيرها كثير.

٣ - **لغات البرمجة** Programming Languages

وهذه اللغات هي التي تستخدم في بناء البرامج المختلفة وهي تتراوح من اللغات التي تتعامل مباشرة مع المكونات المادية للحاسب والأخرى التي تتطلب تحويلها من صورتها التي تكتب بها إلى صورة أخرى يستطيع الحاسب التعامل معها.

ويوجد العديد من لغات البرمجة المستخدمة اليوم وهذه اللغات يمكن تقسيمها إلى ثلاث أنواع رئيسية هي: -

١ - لغة الآلة **Machine languages**

٢ - لغات التجميع **Assembly languages**

٣ - لغات المستوى العالي **High level languages**

لغة الآلة Machine Language

وهي اللغة الوحيدة التي يفهمها الحاسب ويستطيع التعامل معها. وهذه اللغة تعتبر لغة خاصة لكل حاسب وقد تختلف من حاسب إلى آخر وهي تعتمد على المكونات المادية للحاسب نفسه، ولغة الآلة تتكون من مجموعة أرقام من بين 0، 1 التي تعطي تعليمات للحاسب للقيام بمعظم العمليات الأساسية واحدة بعد الأخرى، وهي تختلف من حاسب إلى حاسب آخر ولذلك فإننا نجد أن نفس البرنامج الذي يعمل على حاسب معين قد لا يعمل على حاسب آخر يختلف عنه في المكونات المادية. و لغة الآلة من اللغات الصعبة في التعلم للإنسان حتى بالنسبة للمبرمجين لأنها عبارة عن مجموعة من الأرقام (٠، ١) فقط. وللتغلب على هذه الصعوبة تم اقتراح لغة أخرى تعتمد على استخدام اختصارات معبرة من اللغة الإنجليزية للتعبير عن العمليات الأولية التي يقوم بها الحاسب وهذه اللغة هي لغة التجميع.

لغة التجميع Assembly Languages

هي لغة تستخدم اختصارات معبرة من اللغة الإنجليزية لتعبر بها عن العمليات الأولية التي يقوم بها الحاسب مثل إضافة Add و حفظ Store وطرح Sub وغيرها.

مثال على ذلك

```
Load   A
Add    B
Store  C
```

ونظراً لأن هذه اللغة تستخدم كلمات مختصرة من اللغة الإنجليزية فإنها تحتاج محولاً لكي يحولها إلى لغة الآلة وهو ما يسمى المجمع assembler الذي يقوم بتحويل لغة التجميع إلى لغة الآلة كي يفهمها الحاسب ويستطيع تنفيذها ، وبالرغم من تقليل المجهود الملقى على عاتق المبرمج للقيام بعملية البرمجة إلا أنه ما زالت توجد مشقة عند حل أبسط المسائل لأن ذلك يتطلب معرفة وكتابة العديد من التعليمات، وهذا ما دفع المبرمجين للتفكير في لغات أخرى تقلل المجهود الكبير اللازم لكتابة الكثير من التعليمات فكانت لغات البرمجة ذات المستوى العالي.

لغات البرمجة ذات المستوى العالي High Level Languages

وهذه اللغات كتبت بحيث تستخدم بعض الكلمات الإنجليزية العادية بنفس معانيها حيث يقوم كل أمر منها بتنفيذ العديد من الواجبات، وهذه اللغات كسابقتها تحتاج إلى مترجمات Compilers التي تقوم بتحويل التعليمات(الأوامر) إلى لغة الآلة، .وهذه اللغات تستخدم العلاقات والعوامل الرياضية المتعارف عليها. مثال ذلك

$$Sum = A + B + C$$

وهذه اللغات تعتبر سهلة ومرغوبة من وجهة نظر المبرمجين بالمقارنة بلغات التجميع ولغة الآلة وذلك لسهولة كتابتها وفهمها وحل المشاكل باستخدامها ، ومن أمثلة هذه اللغات لغة C ، ++C، الباسكال Pascal، الفورترانFortran، البيسك Basic، الآداADA ،الجافا Java وغيرها.

ومن المعلوم أن عملية تحويل البرنامج من لغة ذات مستوى عال إلى لغة الآلة تستهلك وقتاً ولذلك تم تطوير نسخ من لغات المستوى العالي بحيث تستخدم برنامج مفسرInterpreter والذي يقوم بترجمة الكود سطرا سطرا أثناء التنفيذ.

.وبالرغم من أن البرامج المترجمة الناتجة من عملية الترجمة باستخدام المترجم compiler تكون أسرع في التنفيذ عن البرامج التي تستخدم المفسر (Interpreter) إلا أنه يفضل وجود نسخة من اللغة تعمل باستخدام المفسر وذلك لسهولة التغيير والحذف والإضافة والتصحيح. وبعد الانتهاء من كل التعديلات والوصول إلى نسخة نهائية فإنه يتم استخدام المترجم لترجمة البرنامج وإنتاج نسخة تنفيذية حتى تكون أسرع في التنفيذ بعد ذلك عند تشغيلها على الحاسب.

أهمية مهنة البرمجة

من المعلوم أن الذي يقوم بكتابة البرامج لحل المشكلات الكثيرة والمعقدة هم المبرمجون ولا يمكن الاستغناء عنهم بحال من الأحوال لأن دورهم مهم وحيوي وتكثر الحاجة لهم في شتى المجالات وذلك لعمل الآتي: -

١ - كتابة برامج وبناء الأنظمة المختلفة لحل المشاكل وتبسيط التعامل مع الحاسب.
٢ - المسئولية الكاملة عن إصلاح ما يحدث من أعطال أو حل المشاكل التي تحدث في الأنظمة المختلفة.
٣ - بناء واجهة المستخدم المختلفة في كثير من اللغات والتطبيقات.
٤ - بناء نظم التشغيل المختلفة مثل Unix، Windows وغيرها من النظم. فمثلاً تستخدم لغة C في بناء نظام التشغيل Unix.
٥ - برامج المواجهة المختلفة في الأنظمة المختلطة الرقمية و التماثلية.

صناعة البرمجيات

تعتبر صناعة البرمجيات في عصرنا الحالي من الصناعات المهمة جدا والتي تتطور باستمرار نتيجة التطور الهائل في صناعة الحاسبات الآلية، ولذلك فإن هذه الصناعة تتطلب مبرمجين مهرة ولديهم القدرة على تحليل وحل المشاكل بالإضافة إلى إلمام بكل المستجدات والعلوم والتطوير المتعلق بالحاسب وصناعة الحاسبات و ذلك حتى يستطيعوا مواكبة تطوير البرامج والنظم المختلفة للاستفادة العظمى من التقدم في الحاسبات.

تمارين

1- أكمل العبارات الآتية بكلمات مناسبة

أ - من أمثلة برامج التشغيل.................، ....، ....

ب - تُقَسَّم البرمجيات إلى ثلاثة أنواع رئيسية هي: -

١ - ....

٢ - ....

٣ - ....

ج - يوجد العديد من لغات الحاسب العالية المستوى مثل..............، ....، ....، ....

د - برنامج الحاسب هو عبارة عن .... تُعطَى للحاسب للقيام بعمل ما مثل............، ....

2- ضع علامة (√) أمام العبارة الصحيحة وعلامة (×) أمام العبارات الخاطئة

أ- برامج التشغيل تقوم بدور الوسيط بين المكونات المادية المكونة للحاسب والمسُتخدم ( )

ب- لغة الآلة تعتبر أسهل لغات البرمجة ( )

ت- البرامج المكتوبة بلغة المستوى العالي يتم تنفيذها مباشرةً ( )

ث- الحاسب لا يفهم إلا لغة الآلة ( )

ج- المترجمات تقوم بتحويل لغة البرمجة إلى لغة الآلة ( )

الفصل الثاني

حل المشكلة Problem Solving

الجدارة:

المساعدة في تحليل المشكلة وتخطيط الحل لهذه المشكلة باستخدام خرائط التدفق والخوارزميات

الأهداف:

عندما تكمل هذه الوحدة يكون لديك القدرة على

1- معرفة أجزاء المشكلة الرئيسية والفرعية
2- تحديد الاحتياجات المطلوبة لحل المشكلة
3- المشاركة بوضع ورسم خريطة التدفق للبرنامج
4- المشاركة في كتابة خوارزمية الحل للمشكلة

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة 100٪

الوقت المتوقع للتدريب: 8 ساعات

الوسائل المساعدة:

- قلم
- دفتر

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة

الفصل الثاني

حل المشكلة

Problem Solving

مقدمة

القدرة على حل المشاكل بواسطة البرمجة هي مهارة وطريقة مرتبة ولا تعتمد على العشوائية، وهذه القدرة يمكن اكتسابها وتعلمها باتباع بعض القواعد التي تساعد على ذلك، وبعض هذه القواعد ذكرها رين ديكارت الرياضي والفيلسوف المعروف وهي: -

١ - لا يمكن قبول أي شيء حقيقة مسلمة إلا إذا ثبت ذلك بالتجربة والمشاهدة.
٢ - كل مشكلة أو معضلة يتم تبسيطها وتقسيمها إلى أجزاء عدة كلما أمكن ذلك.
٣ - فكر بطريقة منظمة ومنطقية وذلك بالبدء بالأجزاء البسيطة والسهلة الفهم ثم التدرج إلى الأجزاء الأصعب وهكذا حتى يتم الانتهاء من المشكلة.
٤ - المراجعة لجميع الأجزاء حتى يكتمل الحل.

وبالرغم من أن هذه القواعد تم وضعها قبل ٣٠٠عام من صناعة أول حاسب إلكتروني إلا أنها ما زالت مطبقة وصالحة للاستخدام، والتفكير الجيد والمنظم لتعريف وتحديد المشكلة ضروري ومهم جداً وأساسي للحصول على نتائج صحيحة وبخاصة عند التعامل مع الحاسب، ولذلك فإن أول خطوة لحل المشكلة هو فهمها.

فهم المشكلة

المشاكل دائماً تظهر أكثر تعقيداً عن الحقيقة التي هي عليها وذلك لعدم فهم المشكلة. ومن معالجة القاعدة الأولي لديكارت والتي تنص على التأكد مما تريد يمكن الحصول على القاعدة الأولى لحل المشكلة وهي: -

قاعدة ١

حلل المشكلة بعناية فائقة محاولاً فهم كل جزئياتها وتحديد كل المتطلبات للحصول على الحل المقبول وفهم كل ما يؤدي للحصول على الحل المقبول للمشكلة.

فإذا وجد حل، بين كيف يمكن العمل لتحقيق هذا الحل. ولذا يجب تحديد مستوى النتائج المطلوبة في المراحل الأولى كما يجب أن تكون الأهداف واضحة ومعلومة وكذلك الوسائل اللازمة لتحقيق هذه

الأهداف ، وملخص هذه القاعدة هو أن فهم المشكلة يمثل نصف الحل وكذلك الفهم الجيد والصحيح والكامل للمشكلة يعطي دائماً نتائج واضحة وصحيح.

تقسيم المشكلة

بزيادة فهم المشكلة يزداد تبعاً له وضوح تفصيلات وأبعاد المشكلة، وبالتالي تصبح المشكلة أكثر تفصيلاً وثباتاً ووضوحا، مما يجعل من الصعب التعامل مع كل هذه التفاصيل في نفس الوقت، وهذا يوضح القاعدة الثانية لديكارت والتي تنص على : -

قاعدة٢

"حاول أن تقسم المشكلة إلى أجزاء بسيطة وغير معتمدة على بعضها البعض ثم ركز على كل جزء على حدة" . وفي هذا الإطار يمكن استخدام العديد من الطرق المختلفة لتقسيم المشكلة ، وبذلك يمكن الحصول على القواعد الفرعية التالية من القاعدة الثانية

قاعدة ٢أ

حاول تقسيم المشكلة إلى مجموعة مشاكل (أجزاء) بسيطة متتابعة، وحتى نحصل على الحل الكامل للمشكلة الأصلية بحل المشاكل الفرعية البسيطة الواحدة تلو الأخرى. والغرض من تقسيم المشكلة هو العمل مع جزء واحد فقط وعزل تأثير الأجزاء الأخرى حتى يسهل التعامل معه، ولكن يجب عدم إهمال ما تقوم به الأجزاء الأخرى من المشكلة لأنه لا يمكن أن تكون معزولة نهائياً عن باقي الأجزاء، ومن المؤكد أن بعض أجزاء المشكلة يجب أن ينظر له ويتم التعامل معه أولاً لأن الأجزاء الأخرى تتأثر به أو تعتمد على النتائج التي تنتج منه. وعند حل كثير من المشاكل فإن ذلك يتضمن تكرار التعامل مع بعض الحالات والأوضاع مثل المستهلكين ، نتائج التجارب......الخ، وفي مثل هذه المشاكل (الحالات) يجب التأكيد على كيفية التعامل مع الحالات الفردية. وإذا كان حل أحد هذه المشاكل (المسائل) كافياً وصحيحاً يمكن للمبرمج أن يعيد استخدام هذا الحل لكل المشاكل المشابهة في جميع الحالات.

قاعدة٢ب

إذا كانت المشكلة تتضمن بعض العمليات التي يعاد تكرارها حاول عزل العمليات التي لا تتطلب الإعادة من تلك التي تتطلب الإعادة.

إذا كنت لا تستطيع أن تقرر من أين تبدأ فإن هذا يحدث لوجود بعض الحالات الخاصة التي تسبب إزعاجاً عند فصلها. وفي هذه الحالة يكون من المفيد أن يتم إهمال هذه الحالات الخاصة وكذلك

الحالات غير المفيدة وغير النافعة في البداية ثم في نهاية الحل يمكن التعامل مع جميع الحالات بما فيها الحالات الخاصة وذلك بعد إجراء بعض التعديلات البسيطة على الحل المقترح.

قاعدة٢ج

في البداية حاول إيجاد حل للمشاكل في الحالات البسيطة أو الحالات المشهورة وعند الوصول إلى حل مرضٍ وصحيح يمكن تطوير هذا الحل ليشمل الحالات الخاصة والمعقدة.

ومن هذه القاعدة نستنتج أن التعامل مع الحالات البسيطة والمشهورة وعند الحصول منها على نتائج مرضية فإن ذلك يشجع على إمكانية الوصول إلى حل للحالات الخاصة. وأما إذا لم نستطيع الحصول على نتائج في الحالات البسيطة فلن نستطيع الحصول على نتائج صحيحة في الحالات الخاصة والمعقدة. ونلخص ذلك بأن تبدأ بالتعامل مع الأجزاء البسيطة ثم تتدرج إلى الأصعب فالأصعب وهكذا.

عملية حل المشاكل

القواعد المؤدية للحل يمكن أن تطبق بطرق مختلفة، كما أنها يجب أن تطبق ببطء وعناية وهذا ما توضحه القاعدة الثالثة

قاعدة٣

"عند تقسيم المشكلة إلى أقسام صغيرة يجب أن يكون التقسيم على خطوات متعددة بحيث تستخدم القواعد العامة في المراحل الأولى ثم يتم الانتقال إلى المراحل الخاصة بعد ذلك"

المراحل الأولى في الحل تتطلب اعتبارات عامة وواسعة بينما المراحل المتأخرة تتطلب التركيز على التفاصيل والانتقال من العام إلى الخاص وهذا ما يعرف بطريقة من الأعلى إلى الأسفل top-down design. ويقترح ألا يتجاوز عدد الأجزاء المقسمة في كل خطوة ٥ أجزاء. والقاعدة الأساسية في عملية التقسيم هي أن يستمر التقسيم حتى يمكن عزل الأجزاء عن بعضها البعض، وأن يكون حل هذه الأجزاء سهلاً. والقدرة على التقسيم تتطلب مهارة عالية وخبرة إلا أن هذه الخبرة يمكن اكتسابها وتطويرها وتنميتها.

قاعدة ٤

"في كل مرحلة من المراحل يجب مراجعة الحل المقترح ليتم التأكد من أنه كامل وصحيح"

يعني ذلك أن مراجعة واحدة للحل لن تكون كافية ويجب تطبيق القاعدة الرابعة عند كل مرحلة. بعد حل واحد من البرامج الفرعية أو الأجزاء يجب إعادة النظر في الحل المقترح لنرى إذا كان يحقق المطلوب

بدقة من هذا البرنامج الفرعي ، وعند تجميع حلول البرامج الفرعية يجب التأكد من التوافق بين كل هذه الحلول للبرامج الفرعية والتأكد من أنها تحقق المطلوب وأنها تأخذ في الحسبان كل الحالات الخاصة. وأخيراً لا تتردد في مراجعة الحلول المقترحة فإنك سوف تجد شيئاً ما يجب أن يضاف أو يعدل أو يحذف.............الخ.

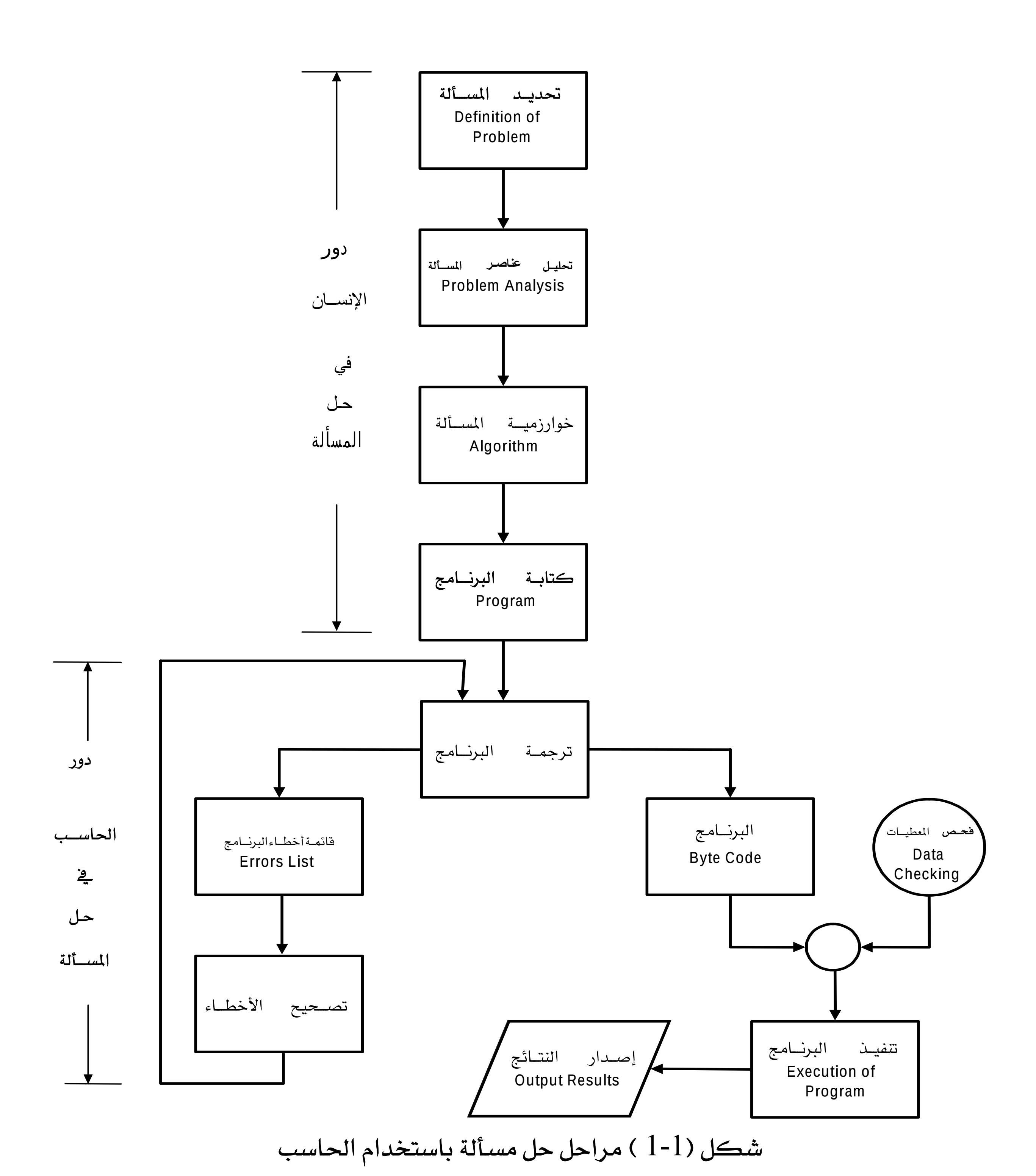
الخوارزم والكود الزائف Algorithm and Pseudo Code

بعد أن استعرضنا خطوات التفكير لحل أية مسألة برمجية وقبل أن ندخل في تفاصيل كتابة الخوارزم لحل المسألة نقول أن الحل يمر بمرحلتين كما هو مبين بشكل (١-1).

المرحلة الأولى

هذه المرحلة تمثل دور الإنسان في حل المسألة وتتكون من عدة خطوات تعرضنا لها فيما سبق ونجملها فيما يأتي:

- تحديد معالم المسألة
- تحليل عناصرها، وذلك بمعرفة معطياتها، والهدف الأساسي لها، وأهم النتائج المطلوبة منها، وما هي الصورة المراد عرض النتائج فيها، وكذلك صورة تقديم المعطيات.
- البحث والتفكير في طريقة حل المسألة
- تدوين الحل في خطوات متسلسلة متعاقبة، يعبر عنها باللغة العادية محكومة بالمنطق الرياضي. هذه الخطوات في مجموعها تسمي بالخوارزم Algorithm، كما يمكن تمثيل هذه الخطوات والارتباط فيما بينها بما يعرف بخريطة التدفق flowchart ، وذلك لكي تساعد في تسلسل المنطق العام لحل المسألة – وسوف نتعرض بالتفصيل لشرح كل من الخوار زم وخرائط التدفق لاحقا في هذا الفصل..
- كتابة البرنامج



شكل (1-1) مراحل حل مسألة باستخدام الحاسب

المرحلة الثانية

وهذه المرحلة تمثل دور الحاسب نفسه في حل المسألة، والتي تبدأ بترجمة البرنامج المكتوب بلغة المستوى العالي إلى لغة الآلة بواسطة المترجم Compiler، ومن ثم يقوم بحفظ البرنامج في الصورة الجديدة حتى يتم تنفيذه بعد ذلك لإخراج النتائج إلى الوسط الخارجي، ليقوم المستخدم بالاستفادة منها بالشكل الذي يريده وذلك عند عدم وجود أخطاء في البرنامج. أما في حالة وجود أخطاء في البرنامج فإنه يجب تصحيح هذه الأخطاء أولاً ثم تعاد الترجمة مرة ثانية وهكذا حتى نحصل على برنامج بدون أخطاء ثم بعد ذلك يتم تنفيذ البرنامج.

الخوارزميات (Algorithms)

لقد استخدمت كلمة الخوارزمية، في القرن الماضي، وبشكل واسع، في أوروبا وأمريكا، وكانت تعني، الوصف الدقيق لتنفيذ مهمة من المهمات، أو حل مسألة من المسائل. وقد اشتق الغربيون هذه الكلمة من اسم عالم الرياضيات المسلم المعروف، محمد بن موسى الخوارزمي.

وتستخدم كلمة الخوارزمية، على نطاق واسع، في علوم الرياضيات والحاسب، الآن حيث تعرف بأنها:
مجموعة الخطوات (التعليمات) المرتبة، لتنفيذ عملية حسابية، أو منطقية، أو غيرها بشكل تتابُعي متسلسل ومنظم.

إن أي خوارزمية تتكون من خطوات مرتبة، بعضها إثر بعض، وكل خطوة تعتبر بنفسها وحدة من وحدات البناء الكامل للخوارزمية، ويختلف حجم هذه الخطوات باختلاف الخوارزميات، واختلاف الأشخاص، الذين يقومون بتنفيذ تلك الخطوات. والمثال التالي يوضح معنى الخوارزمية:
مثال:

إذا أردنا أن نوجد متوسط درجات الحرارة: T_3, T_2, T_1 مثلاً فإن خطوات الحل المنطقية يمكن ترتيبها في الخوارزمية التالية:

الخطوة الأولى: اقرأ قيم درجات الحرارة : T_3, T_2, T_1

الخطوة الثانية: احسب متوسط درجات الحرارة، AV، من المعادلة:

$$AV = (T_1 + T_2 + T_3) / 3$$

الخطوة الثالثة: اطبع النتيجة

مثال آخر:

أراد شخص أن يحسب الزكاة، Z، عن أمواله النقدية، CM، والتي بلغت النصاب الشرعي ، بعد مرور حول قمري عليها ، وهي في حوزته ، فكيف يفعل ؟

الحل: من المعروف أن قيمة الزكاة تحسب بنسبة 2.5%، من قيمة المال البالغ النصاب ، ولذا فإن خطوات الحل يمكن ترتيبها عل النحو التالي:

الخطوة الأولى: اقرأ قيمة ما بحوزته من مال نقدي بالغ للنصاب ، CM

الخطوة الثانية: احسب قيمة الزكاة المستحقة Z ، من المعادلة $Z = .025\ CM$

الخطوة الثالثة: اطبع النتيجة Z

خرائط التدفق Flow charts

تستخدم خرائط التدفق في بيان خطوات حل المسألة وكيفية ارتباطها ببعض، باستخدام رموز اصطلاحية لتوضيح خطوات الحل، وهذه الرموز مبينة بشكل رقم (1-2)

أهمية استخدام خرائط التدفق:

من أهم فوائد استخدام خرائط التدفق قبل كتابة أي برنامج، الأمور الآتية:

١. تعطي صورة متكاملة للخطوات المطلوبة لحل المسائل في ذهن المبرمج، بحيث تمكنه من الإحاطة الكاملة بكل أجزاء المسألة من بدايتها وحتى نهايتها.
٢. تساعد المبرمج على تشخيص الأخطاء التي تقع عادة في البرامج، وبخاصة الأخطاء المنطقية منها ، والتي يعتمد اكتشافها على وضع التسلسل المنطقي، لخطوات حل المسألة لدى المبرمج.
٣. تيسر للمبرمج أمر إدخال أي تعديلات، في أي جزء من أجزاء المسألة، بسرعة، ودون الحاجة لإعادة دراسة المسألة، برمتها من جديد.
٤. في المسائل التي تكثر فيها الاحتمالات والتفرعات، يصبح أمر متابعة دقائق التسلسل، أمراً شاقاً على المبرمج، إذا لم يستعن بمخطط تظهر فيه خطوات الحل الرئيسة بشكل واضح.

	معنى الرمز	(الرمز) الشكل الإصطلاحي
(1)	بداية أو نهاية البرنامج (STRRT / STOP)	
(2)	إدخال أو إخراج (INPUT / OUTPUT)	
(3)	عمليات حسابية وتخزين (CALCULATION AND STORE)	
(4)	تقرير (DECISON)	
(5)	تكرار أو دوران (LOOPING)	
(6)	استدعاء برنامج فرعي (CALL SUBROUTINE)	
(7)	اتجاه سير البرنامج (FLOW LINE)	
(8)	نقطة توصيل وربط (CONNECTOR)	
(9)	تعليق وإيضاح (COMMENT)	

شكل(2-1) الرموز الاصطلاحية لخرائط التدفق

٥. تعتبر رسوم خرائط التدفق المستعملة في تصميم حلول بعض المسائل، مرجعاً، في حل مسائل أخرى مشابهة، ومفتاحاً لحل مسائل جديدة لها علاقة مع المسائل القديمة المحلولة، فتُشبَّه رسوم خرائط التدفق، والحالة هذه، بالرسوم التي يضعها المهندس المعماري عند تصميمه بيتاً أو عمارة، أو مسجداً.... الخ.

أنواع خرائط التدفق

بشكل عام، يمكن القول بأن هناك نوعين، رئيسين من خرائط العمليات وهما:

أ) خرائط سير النظم **System Flowcharts**

يستخدم هذا النوع من الخرائط عند تصميم الأجهزة الهندسية، في المصانع وغيرها، والتي تستعمل أنظمة تحكم ذاتية، مثل العوامة في خزانات المياه، وإشارات المرور الضوئية، وأجهزة ضبط الضغط ودرجات الحرارة في أبراج تقطير البترول، فتعتبر خرائط التدفق هنا، بمثابة المخطط الكامل الذي يبين ترتيب، وعلاقة، ووظيفة، كل مرحلة بما قبلها، وبما بعدها، داخل إطار النظام المتكامل، ويمكن تلخيص الدور الذي تقدمه هذه الخرائط بما يأتي:

١ - تبين موقع كل خطوة من الخطوات الأخرى المكوِّنة للنظام، بحيث يسهل اكتشاف أي خلل يحدث في النظام كله بمجرد النظر، مما ييسر عمليات صيانة الأجهزة، و بأقل التكاليف.
٢ - تسهل إجراء التعديلات التي قد تطرأ مستقبلاً على برنامج النظام في أي موقع منه.
٣ - بيان التفصيلات عن المعطيات المطلوب إدخالها إلى النظام.
٤ - بيان التفصيلات عن أنواع النتائج المتوقعة أو المطلوبة من البرنامج المعد للنظام.
٥ - بيان طرق ربط النظام، ببقية الأنظمة الموجودة في المؤسسة المعنية.

ب) خرائط سير البرامج **Programs Flowchart**

ويستعمل هذا النوع من الخرائط، لبيان الخطوات الرئيسة، التي توضع لحل مسألة ما، وذلك بشكل رسوم اصطلاحية، تبين العلاقات المنطقية، بين سائر خطوات الحل، وموقع ووظيفة كل منها في إطار الحل الشامل للمسألة.

هذا، ويمكن تصنيف خرائط سير البرامج هذه إلى أربعة أنواع رئيسة هي:

١ - خرائط التتابع البسيط **Simple Sequential Flowcharts**

٢ - الخرائط ذات الفروع **Branched Flowcharts**

٣ - خرائط الدوران الواحد **Simple – Loop Flowcharts**

٤ - خرائط الدورانات المتعددة **Multi – Loop Flowcharts**

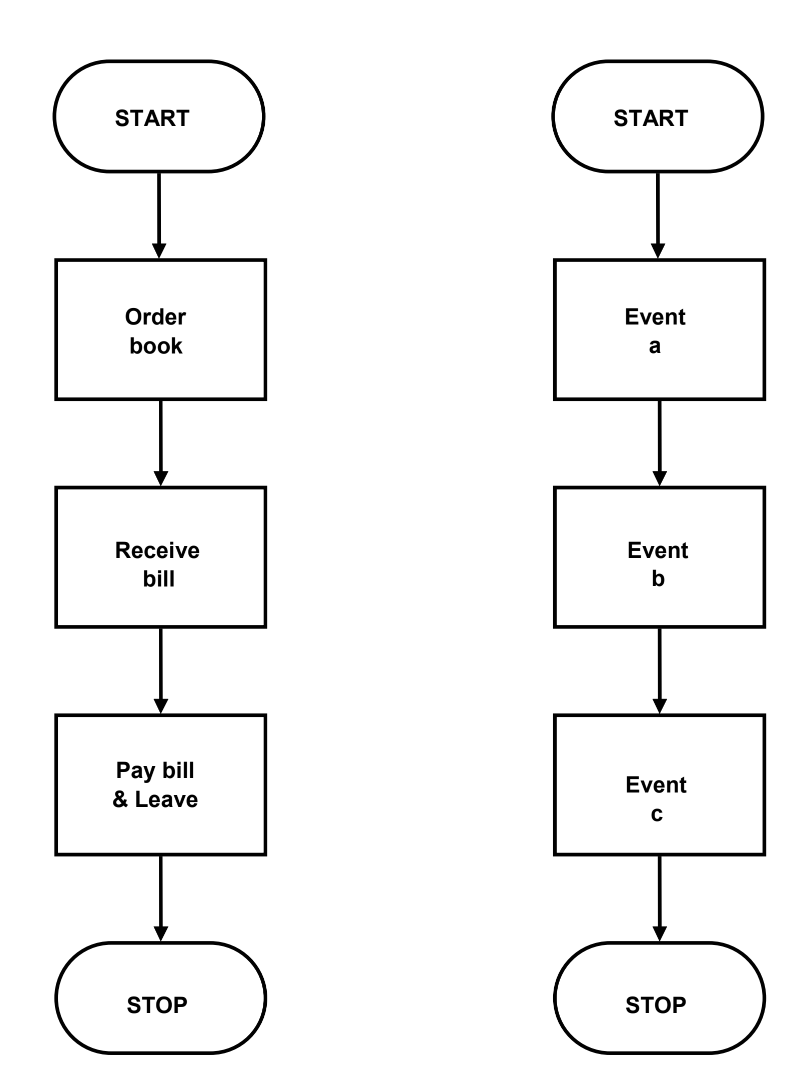
ويمكن للبرنامج الواحد أن يشمل أكثر من نوع واحد من هذه الأنواع، ونتناول فيما يأتي شرح هذه الأنواع بالتفصيل.

خرائط التتابع البسيط

ويتم ترتيب خطوات الحل لهذا النوع من الخرائط، بشكل سلسلة مستقيمة، من بداية البرنامج حتى نهايته، بحيث تنعدم فيها أية تفرعات على الطريق، كما تخلو من أي دورانات مما هو موجود في الأنواع الأخرى من الخرائط. ويكون الشكل العام لهذا النوع كما هو مبين في الشكل (3-1) ، وفيها يتم تنفيذ الحدث a ثم يليه تنفيذ الحدث b وبعده التوقف.
وكلمة الحدث a، الواردة في شكل (3-1) تعني الحدث أو العملية المطلوب تنفيذها.

مثال(1):
ارسم خريطة سير البرنامج التي تمثل عملية شراء كتاب من مركز بيع الكتب.
الحل:
خريطة سير البرنامج في الشكل(4-1) يمكن أن تمثلها الخطوات الآتية:

1- اطلب الكتاب
2- استلم الفاتورة
3- ادفع الفاتورة وغادر



شكل (1-3) خرائط التتابع البسيط	شكل (1-4)

مثال ٢

ارسم خريطة سير البرنامج (flow chart) لإيجاد مساحة ومحيط دائرة نصف قطرها معلوم (R)

الحل:

مساحة الدائرة = πR^2

محيط الدائرة = $2\pi R$

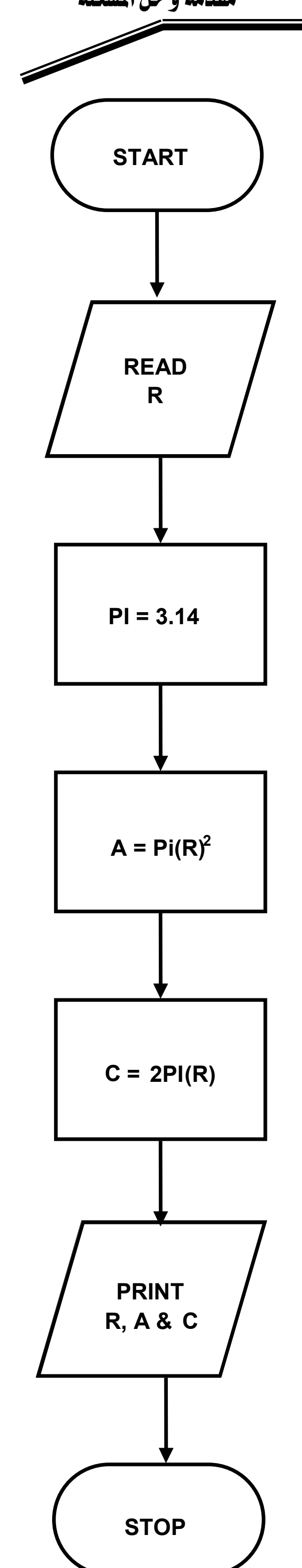
حيث π = النسبة التقريبية وقيمتها العددية ثابتة وتساوي ٣,١٤

بينما R متغير يمثل نصف قطر الدائرة

وحل هذه المسألة كما يأتي :

١- اقرأ قيمة R
٢- ضع قيمة π = pi= ٣,١٤
٣- احسب مساحة الدائرة A من المعادلة $A = \pi R^2$
٤- احسب مساحة المحيط C من المعادلة $C = 2\pi R$
٥- اطبع قيم كل من : A, R, C

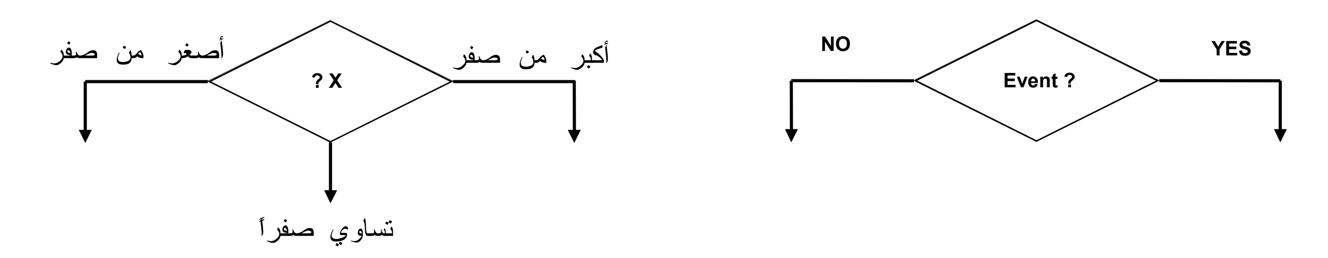
خريطة سير البرنامج التي توضح حل هذه المسألة مبينة في شكل (1-5)



شكل(1-5)

الخرائط ذات الفروع

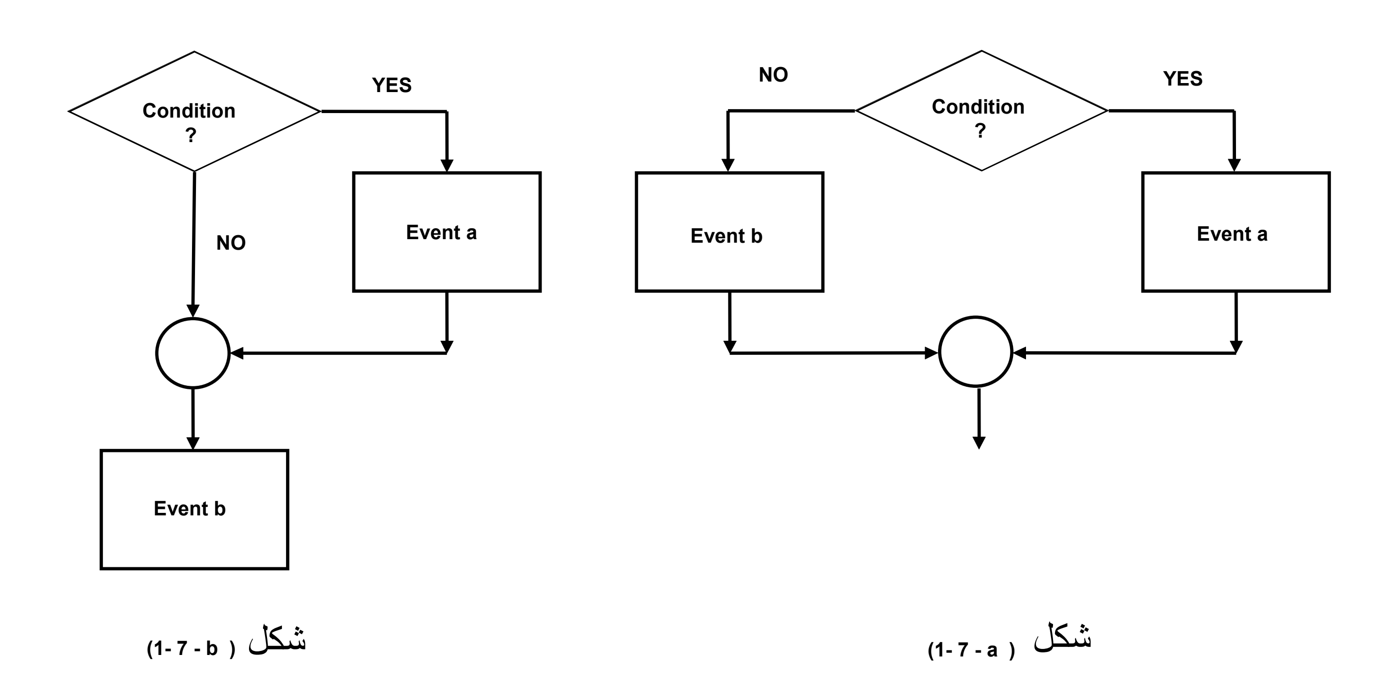
إن أي تفرع يحدث في البرنامج، إنما يكون بسبب الحاجة لاتخاذ قرار، أو مفاضلة بين اختيارين أو أكثر، فيسير كل اختيار في طريق مستقل (تفرع) عن الآخر. وهناك لونان من القرار يمكن للمبرمج استعمال أحدهما حسب الحالة التي يدرسها، والشكل (1-6) يبين هذين المسارين من القرار.



شكل (1-6-a) قرار ذو فرعين

شكل (1-6-b) قرار ذو ثلاثة أفرع

وبشكل عام فإن خرائط التفرع يمكن أن تأخذ إحدى الصورتين الآتيتين كما هو موضح بشكل (1-7). في شكل (1-7-a) يمكن ملاحظة أنه اذا كان جواب الشرط: نعم فإن الحدث التالي في التنفيذ يكون الحدث (a). أما اذا كان الجواب : لا ، فإن الحدث التالي يكون الحدث (b). أما في الشكل (1-7-b) فإننا نلاحظ أنه اذا كان جواب الشرط : نعم، فإن الحدث التالي في التنفيذ يكون الحدث (a) ثم يتبعه الحدث (b).، أما اذا كان جواب الشرط :لا ، فإن الحدث التالي يكون الحدث (b) مباشرة



شكل (1-7-a)

شكل (1-7-b)

مثال ٣

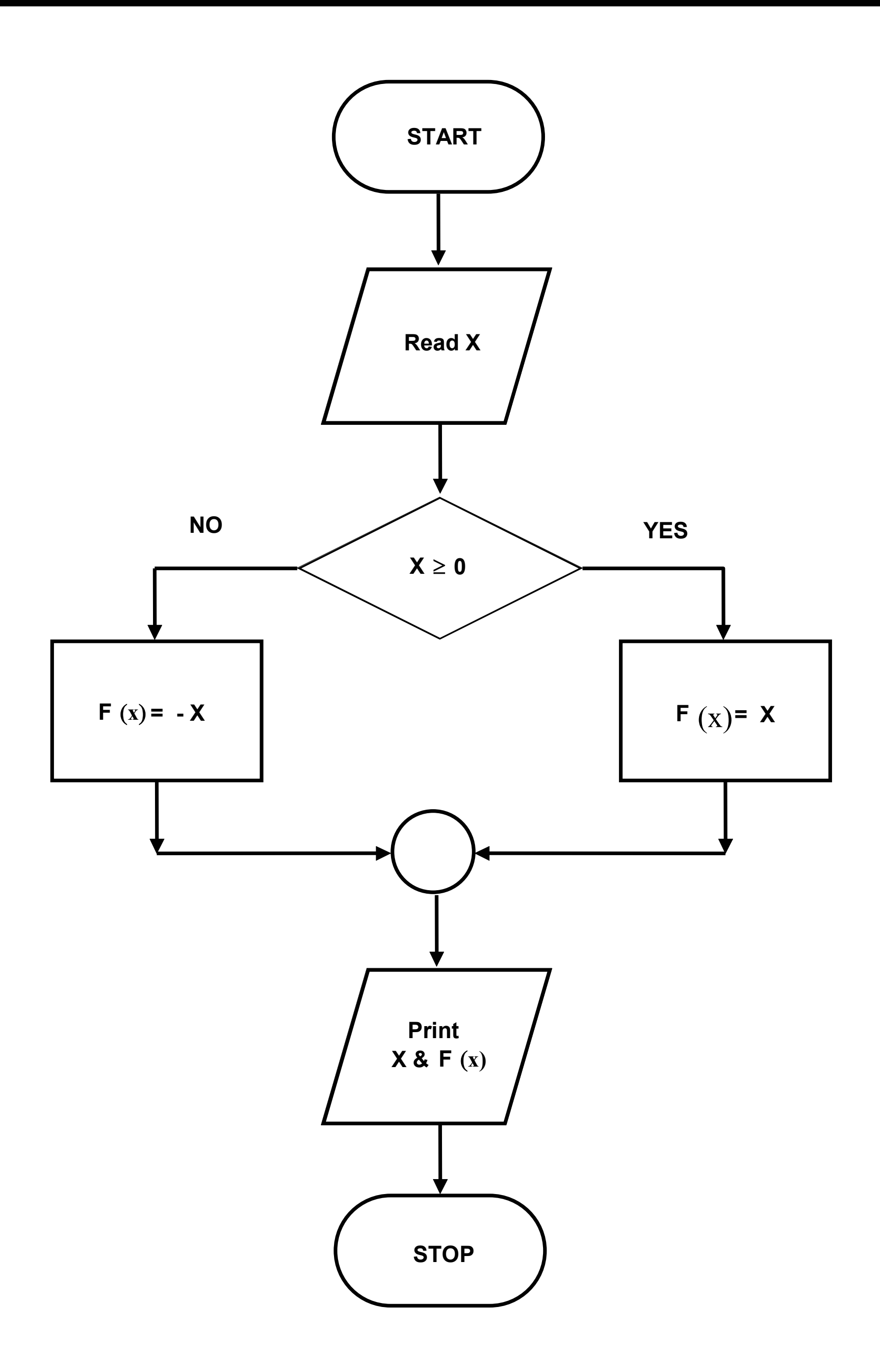
ارسم خريطة سير البرنامج (flow chart) لإيجاد قيمة الدالة F(x) المعرف كما يلي :

$$F(x) = \{ \begin{matrix} x & x \geq 0 \\ -x & x \leq 0 \end{matrix}$$

الحل:

شكل (1-8) يبين خريطة سير البرنامج لحل هذه المسألة كما يلي:

١ - اقرأ قيم المتغير X
٢ - اذا كانت X أكبر من أو تساوي صفرا اذهب إلى خطوة ٣، وإلا فاذهب إلى الخطوة ٤
٣ - احسب قيمة الدالة F(x) من F(x) =X ثم اذهب إلى الخطوة ٥
٤ - احسب قيمة الدالة F(x) من F(x)
٥ - اطبع قيم كل من X , F(x)



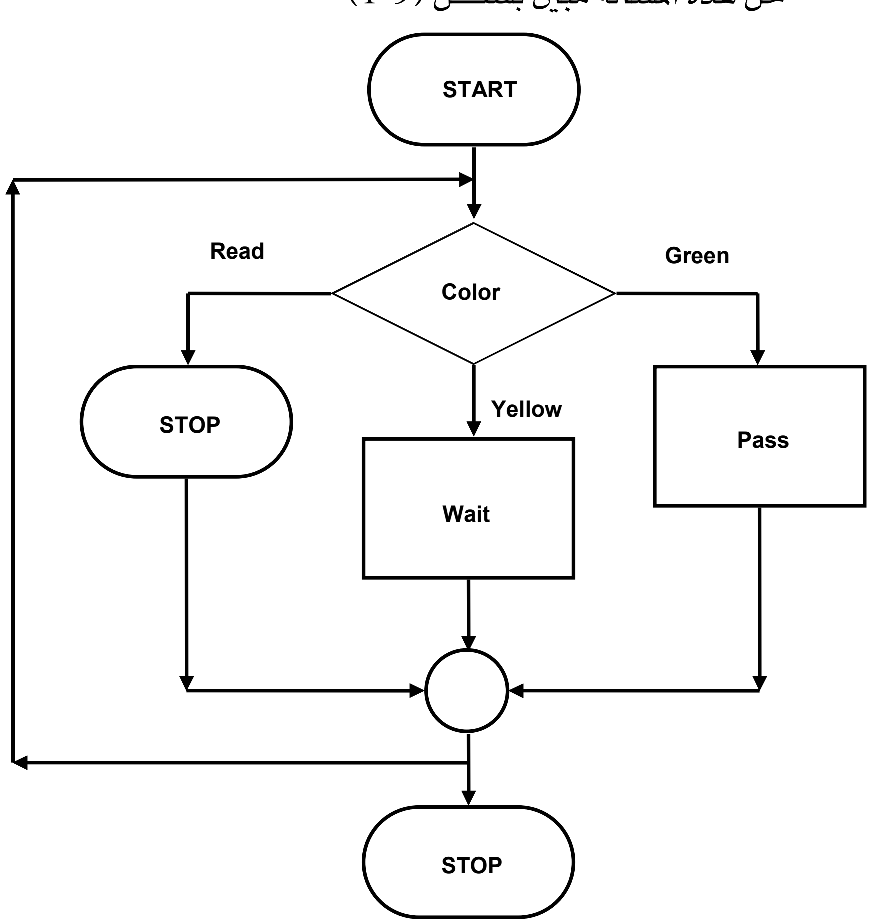
شكل (1-8)

مثال ٤

ارسم خريطة سير البرنامج لإشارات السير الضوئية (اشارات المرور)

الحل:

حل هذه المسألة مبين بشكل (1-9)



شكل (1-9)

مثال ٥

ارسم خريطة سير البرنامج لحساب قيمة W ،
من المعادلات الآتية علماً بأن قيمة المتغير X
معلومة

$$W = \begin{cases} X+1 & \text{if } x > 0 \\ \sin(x)+5 & \text{if } x = 0 \\ 2X-1 & \text{if } < 0 \end{cases}$$

الحل :

خطوات الحل مبينة في شكل
(1-10) وهي:

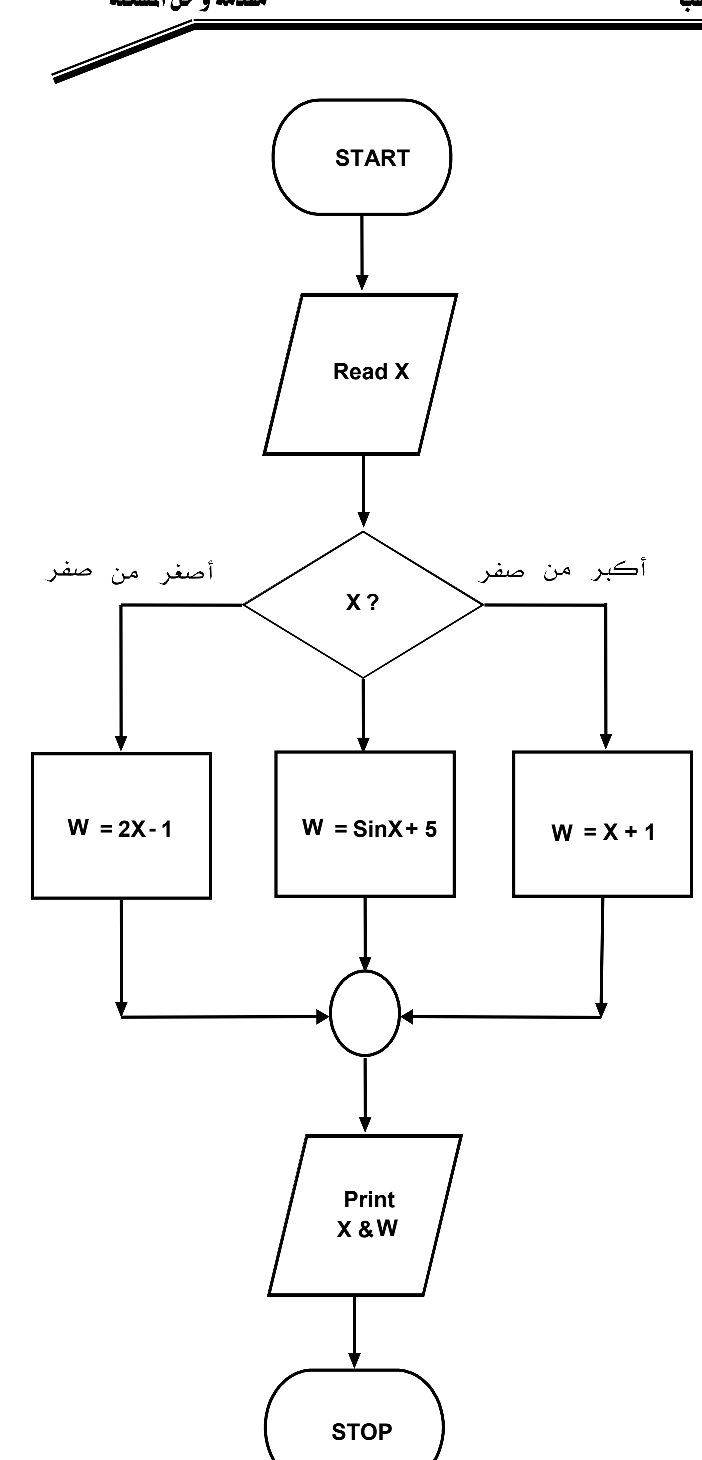
١- إذا كانت X أكبر من الصفر اذهب إلى
الخطوة ٢
إذا كانت X تساوي صفرا اذهب إلى الخطوة ٣
أما إذا كانت X أصغرمن الصفر اذهب إلى
الخطوة ٤

٢ - احسب W من المعادلة W= X+1 ثم
اذهب إلى الخطوة ٥

٣ - احسب W من المعادلة W= sin (X) +5
ثم اذهب إلى الخطوة ٥

٤ - احسب W من المعادلة W=2X -1

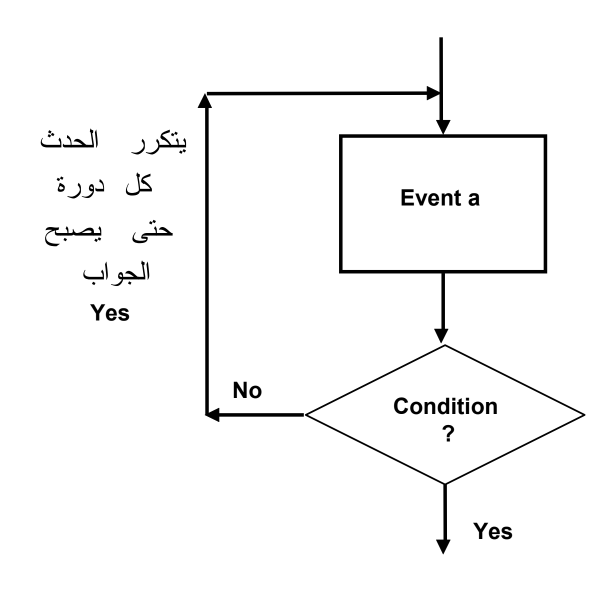
٥ - اطبع قيم كل من X, W

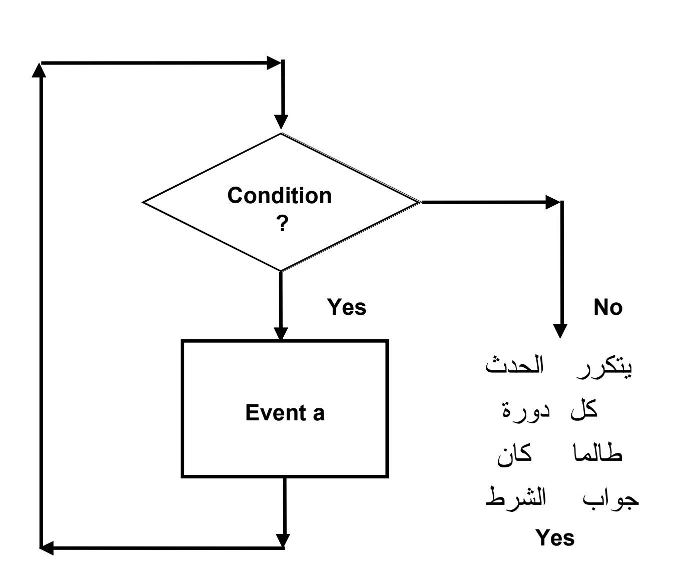


شكل (1-10)

خرائط الدوران الواحد:

وهذه الخرائط نحتاج اليها لإعادة عملية أومجموعة من العمليات في البرنامج عددا محدودا أو غير محدود من المرات، والشكل العام لمثل هذه الخرائط مبين بشكل (1-11). وقد سميت هذه الخرائط بخرائط الدوران الواحد لأنها تستعمل حلقة واحدة، وتسمى أحيانا خرائط الدوران البسيط،





شكل(1-11)

مثال ٦

ارسم خريطة سير البرنامج لإيجاد مساحة مجموعة من الدوائر أنصاف أقطارها معلومة

الحل : خطوات الحل مبينة في شكل (1-12) وهي:

١ - اقرأ نصف قطر الدائرة R

٢ - أوجد مساحة الدائرة A

٣ - اطبع قيم كل من A,R

٤ - هل هناك المزيد من الدوائر ؟

إذا كان نعم عد للخطوة ١

أما إذا كان لا فتوقف

مثال ٧

من واجبات بيت مال المسلمين أن يساعد الشباب على الزواج وذلك بتقديم الدعم المادي المناسب لهم (٣٠٠٠٠ ريال للزيجة الواحدة) على فرض أن السن المناسب للزواج هو عشرون عاما ، اقترح خريطة لسير البرنامج لهذا المشروع.

الحل: خطوات الحل مبينة في شكل (1-13) وهي:

١- خذ شابا على الدور

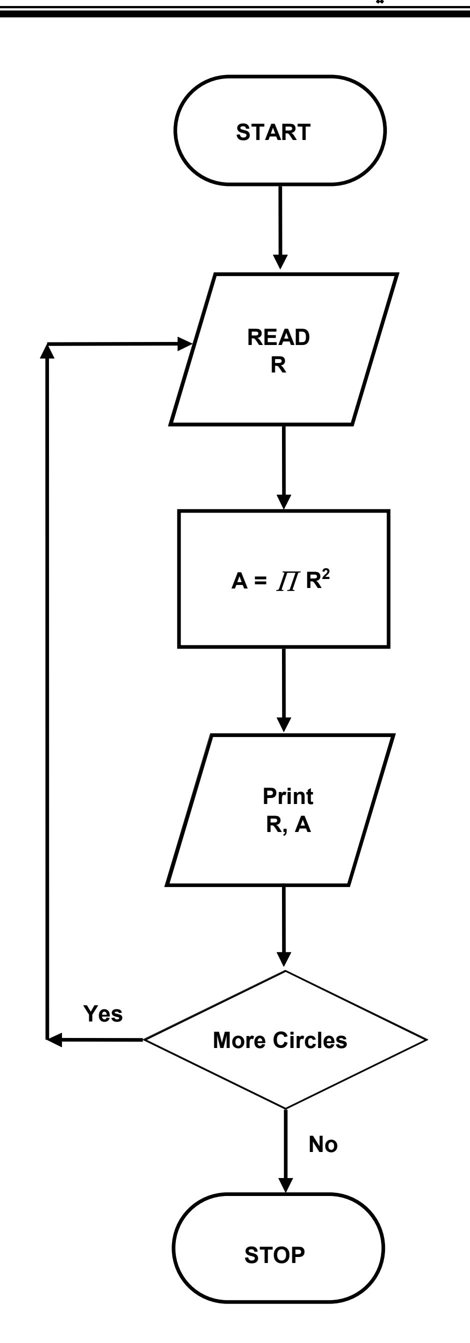
٢- هل مضى من عمره عشرون عاما؟
إن كان نعم اذهب الى الخطوة ٣
أما إن كان لا ، اذهب إلى الخطوة ٤

٣- زوج الشاب المذكور

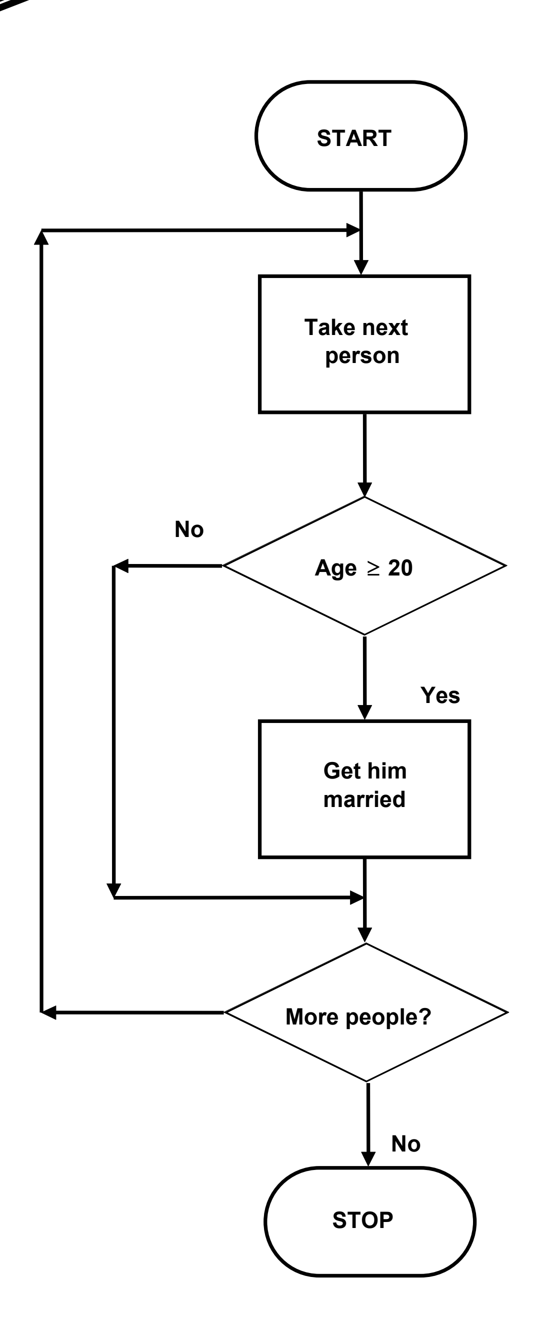
٤- هل هناك مزيد من الشباب؟
إن كان نعم اذهب الى الخطوة ١
أما إن كان لا ، فتوقف

ملحوظة:

ينبغي التنبيه هنا إلى أن عملية الانتقال من خطوة ٢ إلى الخطوة ٤ – عندما تكون الاجابة "لا" – لا تمثل دورانا أو تكرارا لأن عملية الدوران إنما تتم بالانتقال من خطوة متأخرة إلى خطوة متقدمة عدة مرات لإعادتها ، ولذا فإن هناك دورانا بسيطا واحدا في هذا المثال، ويمثله العودة من خطوة ٤ إلى خطوة ١



شكل (1-12)



شكل (1-13)

مثال ٨

ارسم خريطة سير البرنامج لخزان ُيملأ بالماء ذاتيًّا (أتوماتيكيًّا)، عندما يصبح ارتفاع مستوى الماء فيه أقل من متر.

الحل : من المعلوم أن عملية ملء الخزان تقوم على فكرة وجود العوامة التي تفتح صنبور التغذية ذاتياً عندما يصل ارتفاع الماء حداً معيناً (متراً واحداً في هذا المثال) وتغلق صنبور التغذية عند وصول مستوى الماء في الخزان إلى الارتفاع المطلوب وبالتالي فإن خطوات الحل المبينة في الشكل (14-1) تكون كما يأتي:

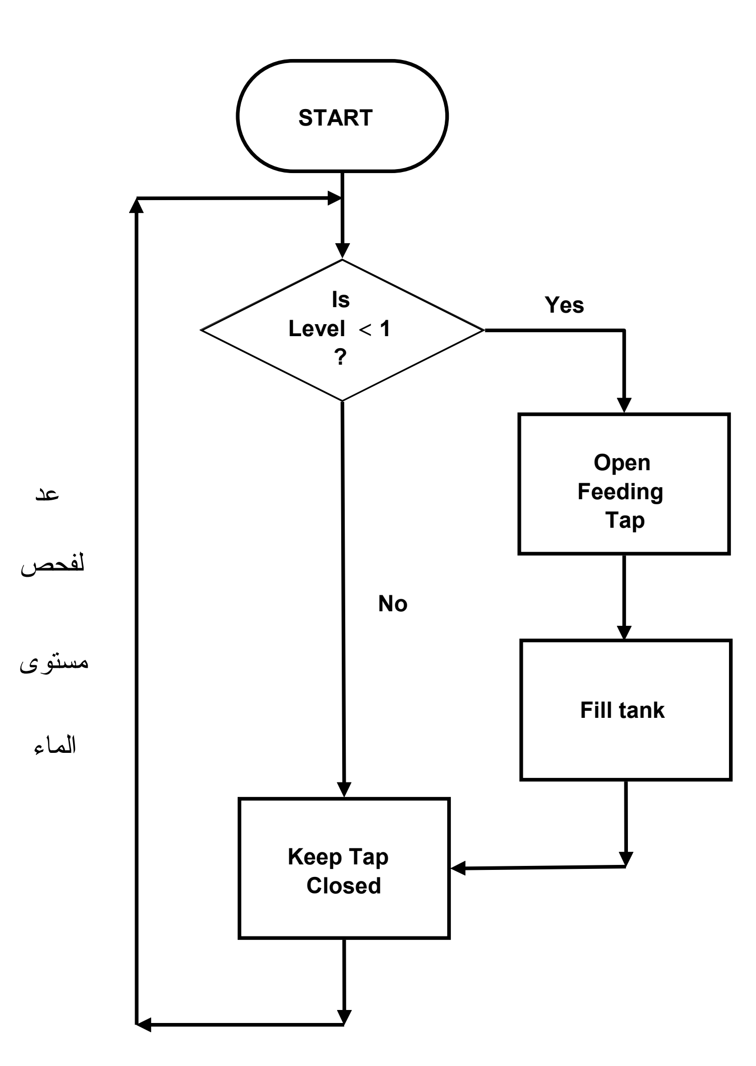
1- هل مستوى الماء أقل من متر ؟ إذا كان الجواب نعم فاذهب إلى الخطوة (2) وإذا كان الجواب لا ، فاذهب إلى الخطوة (4)

2- يفتح صنبور التغذية.

3- يملأ الخزان إلى المستوى المطلوب.

4- أغلق الصنبور (أو حافظ عليه مغلقاً).

5- عد إلى الخطوة (1) لفحص مستوى الماء مرة بعد مرة للحفاظ على الوضع المطلوب وبشكل دائم.



شكل (1-14)

مثال ٩

الشكل (1-15) يمثل خريطة سير البرنامج لمجموعة من العمليات الحسابية. ادرس العمليات بعد تتبع الخريطة.

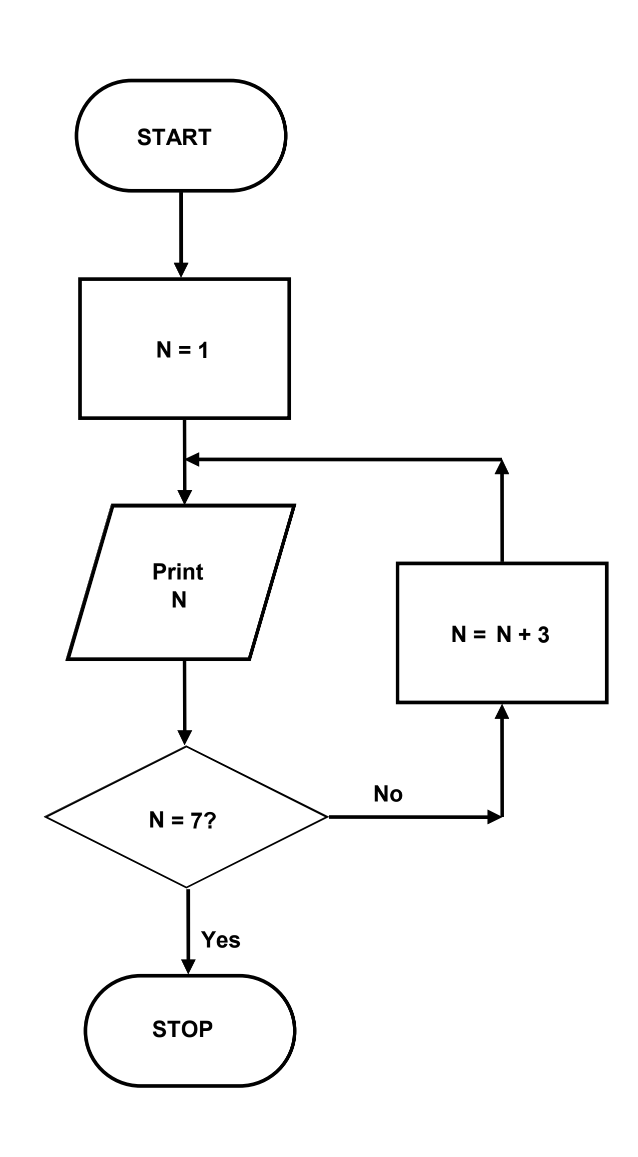
من الشكل نلاحظ أن الحاسب يبدأ بوضع قيمة مبدئية أولى مقدارها 1 في مخزن الذاكرة N كما في الشكل (1-16a)، ثم يقوم بطبع هذه القيمة، من خلال جهاز الإخراج، وعلى الوسط الخارجي ثم يسأل (هل القيمة المخزونة في مخزن N تساوي 7؟) الجواب بالطبع : لا، لأن $7 \neq 1$، لهذا فهو ينفذ الأمر التالي: $N = N + 3$ وهذا الأمر يعني أن الحاسب سيضع في مخزن N ما كان فيه سابقاً مضافاً إليه 3 لتصبح القيمة المخزونة في المخزن N تساوي 4 بدلاً من 1 (انظر الشكل (1-16b)) ثم يعود مرة أخرى بعد أن يطبع N الجديدة على الوسط الخارجي ليسأل هل $7 = 4$ ؟ ويكرر العملية السابقة حتى تصبح القيمة المخزونة في المخزن N تساوي 7 وعندها يتوقف البرنامج بالأمر: توقف، ويكون قد طبع لنا على الوسط الخارجي القيم التي خزنت في مخزن N على التوالي وهي:

1

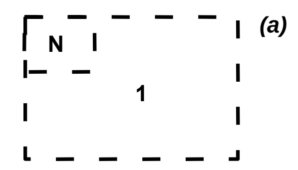
4

7

- ملحوظة: هناك مخزن واحد فقط تحت اسم N في وحدة الذاكرة تخزن فيه قيمة واحدة في الوقت الواحد، ولذا فإن آخر قيمة تبقى في المخزن N في المثال هي 7



شكل (15-1)



(b)
N
4

(c)
N
7

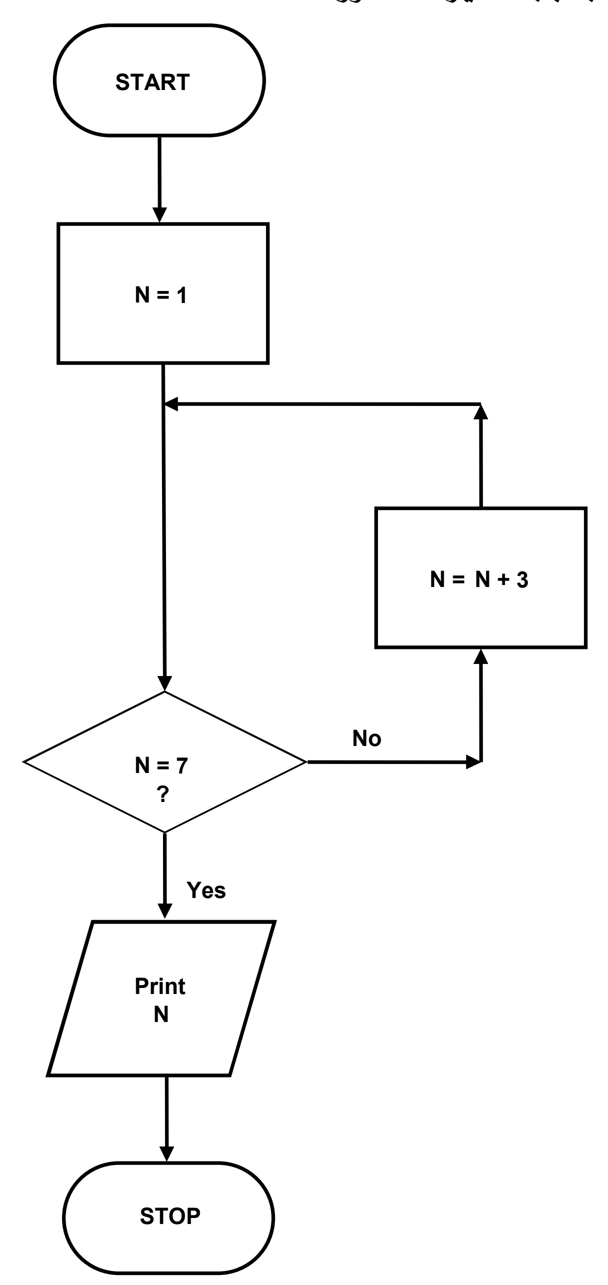
شكل (16-1)

مثال ١٠

لو أحدثنا تغييراً بسيطاً في الشكل (15-1) ليصبح كما هو مبين في شكل (17-1) فما أثر ذلك التغيير؟

نلاحظ من الشكل (15-1) أن التغيير الذي حدث يتلخص في أن خطوة كتابة قيمة المتغير N قد تأخرت عن خطوة التقرير (هل $N = 7$)؟ وهذا يعني أن كتابة قيمة المتغير N تأتي بعد الانتهاء من الدوران أي بعد أن تصبح قيمة N تساوي 7، ولذا فإن نتائج الإخراج تكون قيمة واحدة فقط وهي: 7

في حين أن نتائج الإخراج في المثال السابق كانت تطبع في كل دوران، مما جعل النتائج في المثالين مختلفة بسبب التغير المذكور.



شكل (17-1)

مثال ١١

ارسم خريطة سير البرنامج لإيجاد مجموع m من الأعداد الحقيقية

$(X_1, X_2, ..., X_m)$

الحل: النتيجة المطلوبة هي مجموعة الأعداد T حيث إن $T = \sum_{i=1}^{m} Xi$

خطوات الحل يمكن أن تسير على النحو التالي:

$$T_0 = 0$$
$$T_1 = T_0 + X_1 = 0 + X_1 = X_1$$
$$T_2 = T_1 + X_2 = X_1 + X_2$$
$$T_m = T_{m-1} + X_m = X_1 + X_2 + ... + X_{m-1} + X_m$$

ونموذج الحل هذا يمكن أن يختصر بنموذج مكافىء هو:

$$T_i = T_{i-1} + X_i \qquad (1)$$

حيث $T_0 = 0$ وتتغير i من 1 إلى m

ويمكننا اختصار عدد المتغيرات $T_1 .... T_m$ بمتغير واحد هو T، الذي تكون قيمته في كل دوران مساوية لقيمته السابقة مضافاً إليها قيمة X المراد إضافتها إليه. وذلك بإعادة كتابة المعادلة (1) على النحو:

$$T_i = T + X_i \qquad i = 1, m \qquad (2)$$

على أن تكون القيمة الأولى للمجموع T تساوي صفراً.

وترتيب النموذج في المعادلة (2) من شأنه أن يوفر أكبر عدد ممكن من المخازن الشاغرة في الذاكرة، لاستخدامها في أغراض أخرى، وكذلك فإن صيغة المعادلة (2) أسهل من صيغة المعادلة (1) وبالتالي فإنها تساعد على تسهيل عملية البرمجة.

أما خريطة سير البرنامج فمبيَّنة في شكل (1-18) (افرض أن قيمة m تساوي 100 هنا)

خرائط الدورانات المتعددة

في هذه الحالة تكون الدورانات داخل بعضها البعض بحيث لا تتقاطع فإذا كان لدينا دوران من هذا النوع (انظر شكل (1-19)) فيسمى الدوران رقم (1) دوراناً داخلياً Inner Loop بينما الدوران رقم (2) دوراناً خارجياً Outer Loop، ويتم التنسيق بين عمل مثل هذين الدورانين، بحيث تكون أولوية التنفيذ للدوران الداخلي.

وقد سميت هذه الخرائط بخرائط الدورانات المتعددة لأنها تستعمل أكثر من حلقة دوران واحدة، وقد تسمى أحياناً بخرائط الدورانات المتداخلة أو المتراكبة أو الضمنية nested، وكل هذه التسمية تؤدي إلى معنى واحد.

شكل (1-18)

شكل(1-19)

مثال ١٢

يرغب تاجر في تقطيع مجموعة من قطع القماش طول كل منها يزيد عن 5 أمتار، إلى قطع صغيرة، طول الواحدة منها يساوي 5 أمتار، ارسم خريطة سير البرنامج لهذا المشروع.

خطوات الحل المبينة في شكل (20-1) هي:

1- خذ قطعة

2- اقطع منها قطعة طولها 5 متر

3- هل المتبقي يزيد عن 5 متر؟

إذا كان الجواب نعم، فاذهب إلى الخطوة (2)

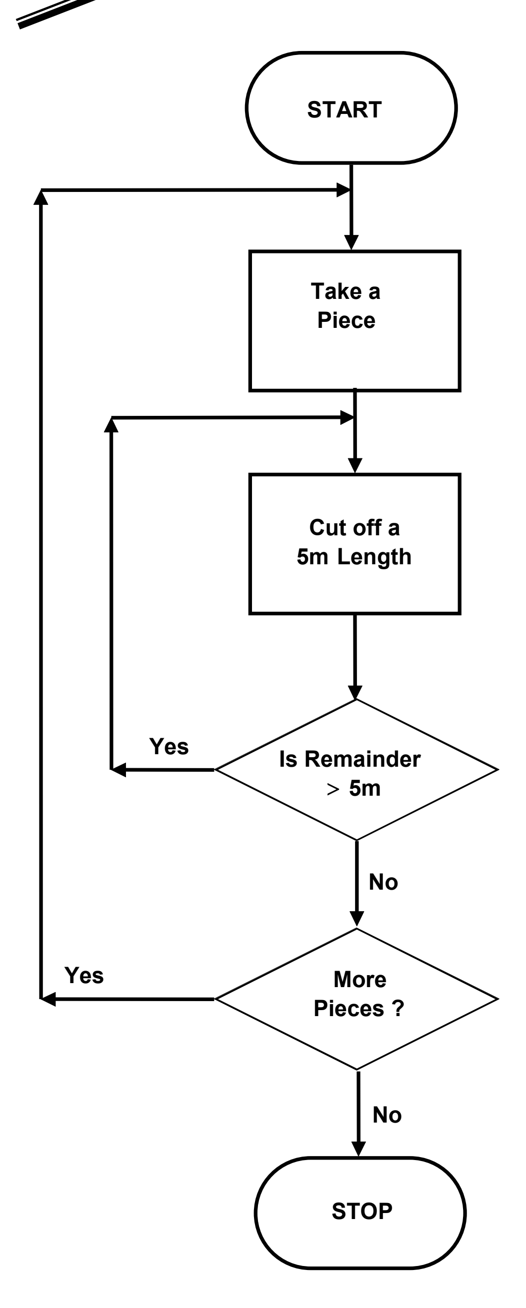
إذا كان الجواب لا ، فاذهب إلى الخطوة (4)

4- هل هناك مزيد من القطع المراد تقطيعها؟

إن كان الجواب نعم، فاذهب للخطوة (1)

وإن كان لا ، فتوقف

- **ملحوظة:** يلاحظ من الشكل (20-1) أن الدوران الداخلي يتضمن تقطيع القطعة الواحدة إلى قطع متعددة، طول كل منها 5 متر، بينما يمثل الدوران الداخلي تناول قطعة واحدة جديدة لتنفذ عليها إجراءات الدوران الداخلي.

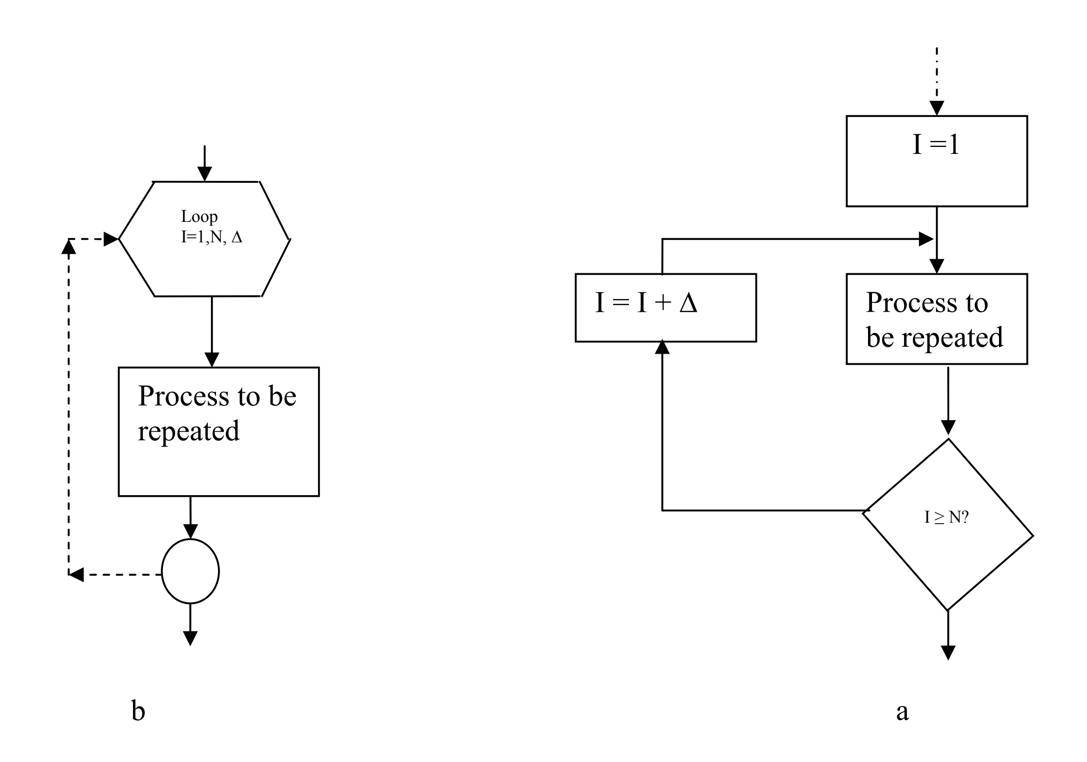


شكل (20)

صيغة الدوران باستعمال الشكل الاصطلاحي

لقد عرفنا في الفقرتين السابقتين مفهوم الدوران البسيط والدورانات الضمنية، ويمكننا الآن استخدام الشكل الاصطلاحي للدوران -الوارد ضمن الرموز الاصطلاحية لخرائط سير البرنامج - على النحو التالي:

نلاحظ في الشكل (1-21) أننا نحتاج إلى العناصر الآتية:



شكل (1-21)

العدّاد(I)

القيمة الأولية للعدّاد I (هنا i = 1)

القيمة النهائية للعدّاد I (هنا N)

قيمة الزيادة في العدّاد عند نهاية كل دورة (Δ)

نلاحظ من الشكل (1-21-a) أن إجراءات الدوران كانت تتم طبقاً للخطوات الآتية والمفصلة من قبل المبرمج:

1- أعط I قيمة أولية.

2- أتم الإجراءات المطلوب إعادتها.

3- اتخاذ قرار: إذا كانت قيمة العداد I وصلت إلى القيمة النهائية N، فاخرج إلى الخطوة التالية في البرنامج وإلا فاذهب إلى الخطوة (4)

4- زد العداد بمقدار الزيادة Δ

5- عد إلى (2)

يمكننا استبدال الخطوات المفصلة (5,4,3,1) في الشكل (1-21-a) بخطوة مجملة واحدة مبينة في الشكل الاصطلاحي للدوران شكل (1-21-b)، حيث تنفذ هذه الخطوات بصورة أوتوماتيكية من قبل الحاسب. وهذا من شأنه تسهيل عملية البرمجة، واختصار عدد العمليات في البرنامج وتجنب بعض الأخطاء.

- ملحوظة: تعتبر قيمة Δ تساوي 1 دائماً إذا لم تُعطَ قيمة أخرى بخلاف ذلك، وفي حالة عدم ذكر قيمة Δ يصبح الشكل الاصطلاحي الوارد في شكل (1-21-b) كما هو موضح بشكل (1-22) وتكون قيمة الزيادة Δ تساوي 1، بصورة أتوماتيكية.

Loop
I = 1, n

شكل (1-22)

مثال ١٣

أعد حل مثال (6) لإيجاد مساحة من الدوائر باستخدام الشكل الاصطلاحي للدوران.

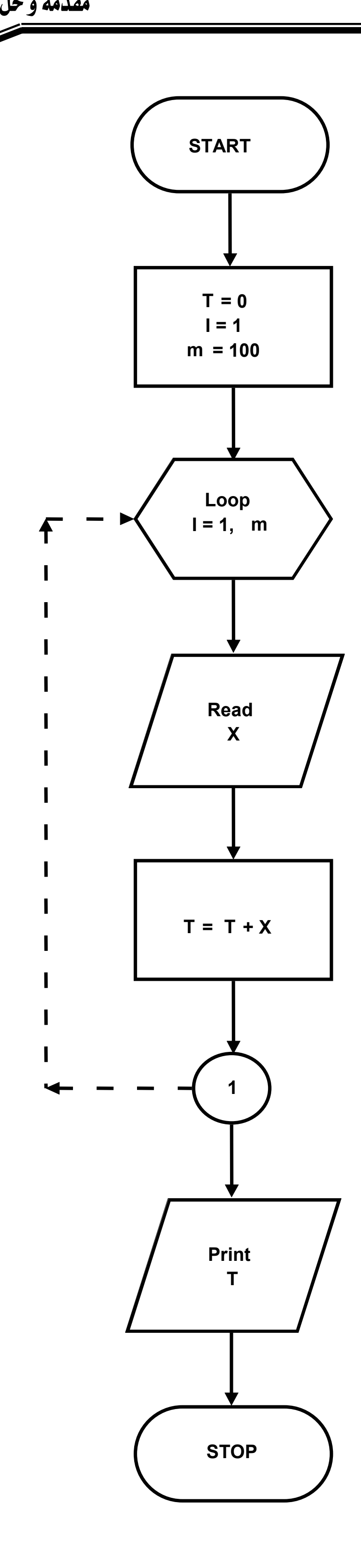
خطوات الحل كما مبينة في الشكل (1-23)

مثال ١٤

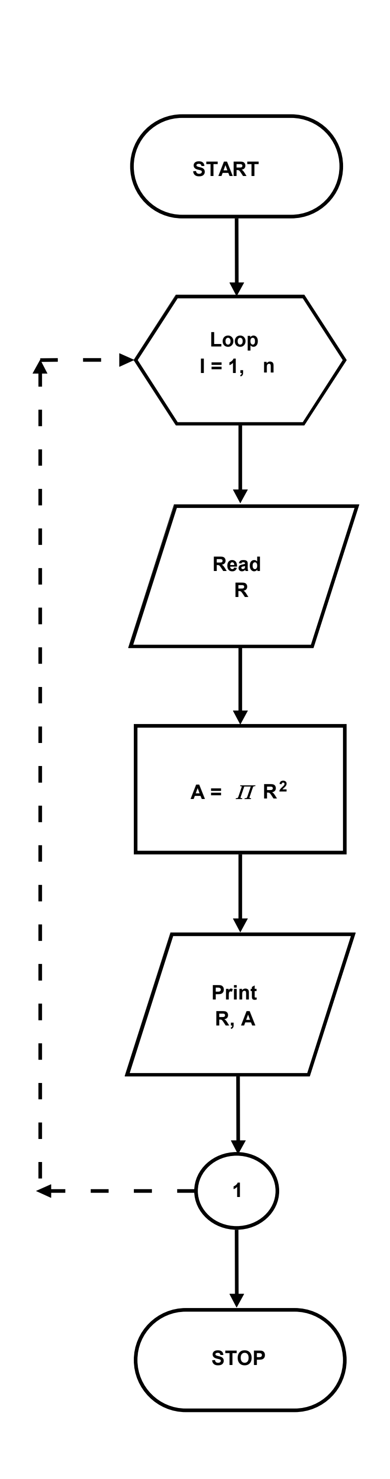
أعد حل مثال (11) باستخدام الشكل الاصطلاحي للدوران. بحيث

m = 100

خطوات الحل كما هي مبينة في الشكل (1-24)



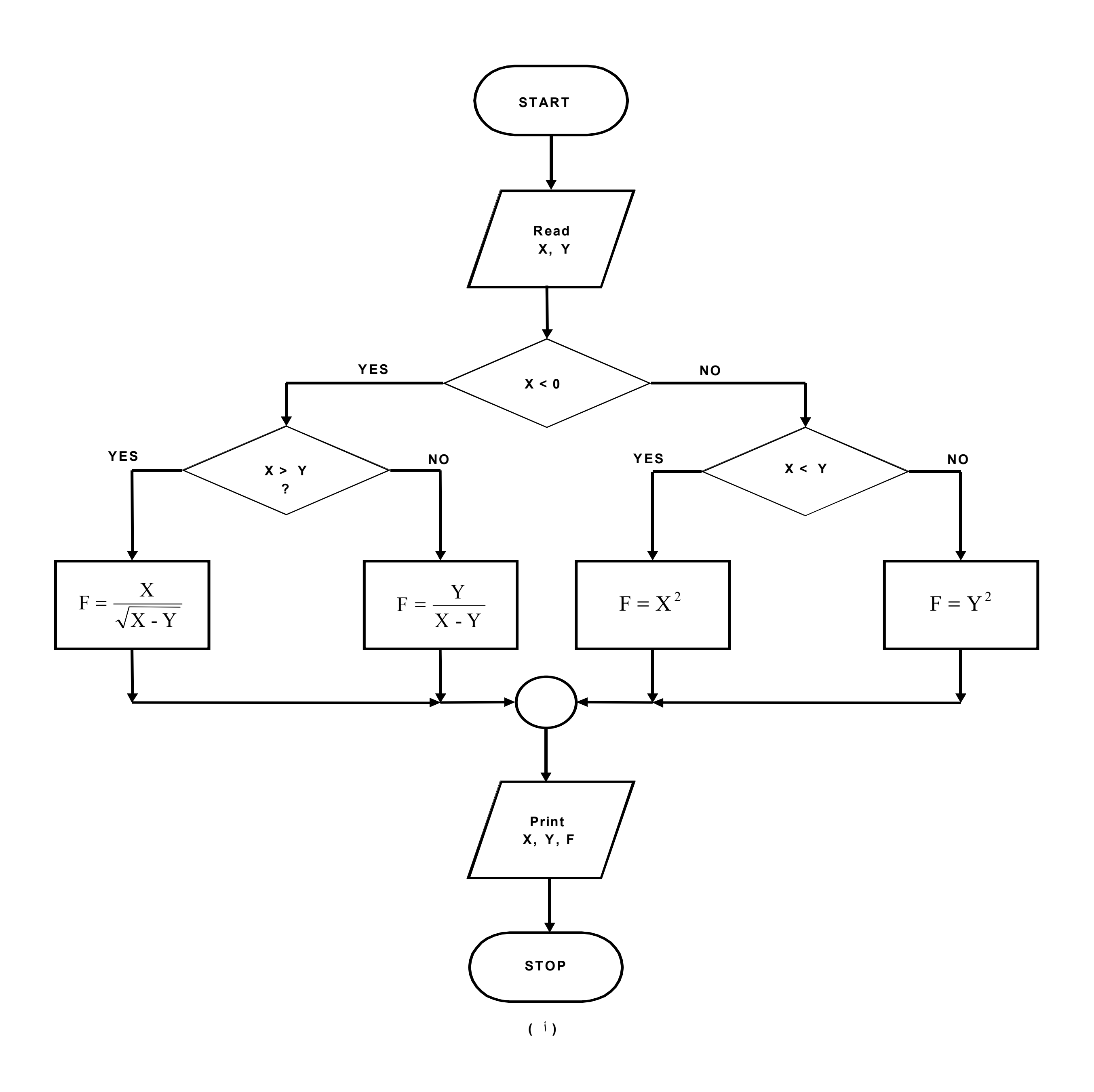
شكل (1-24)



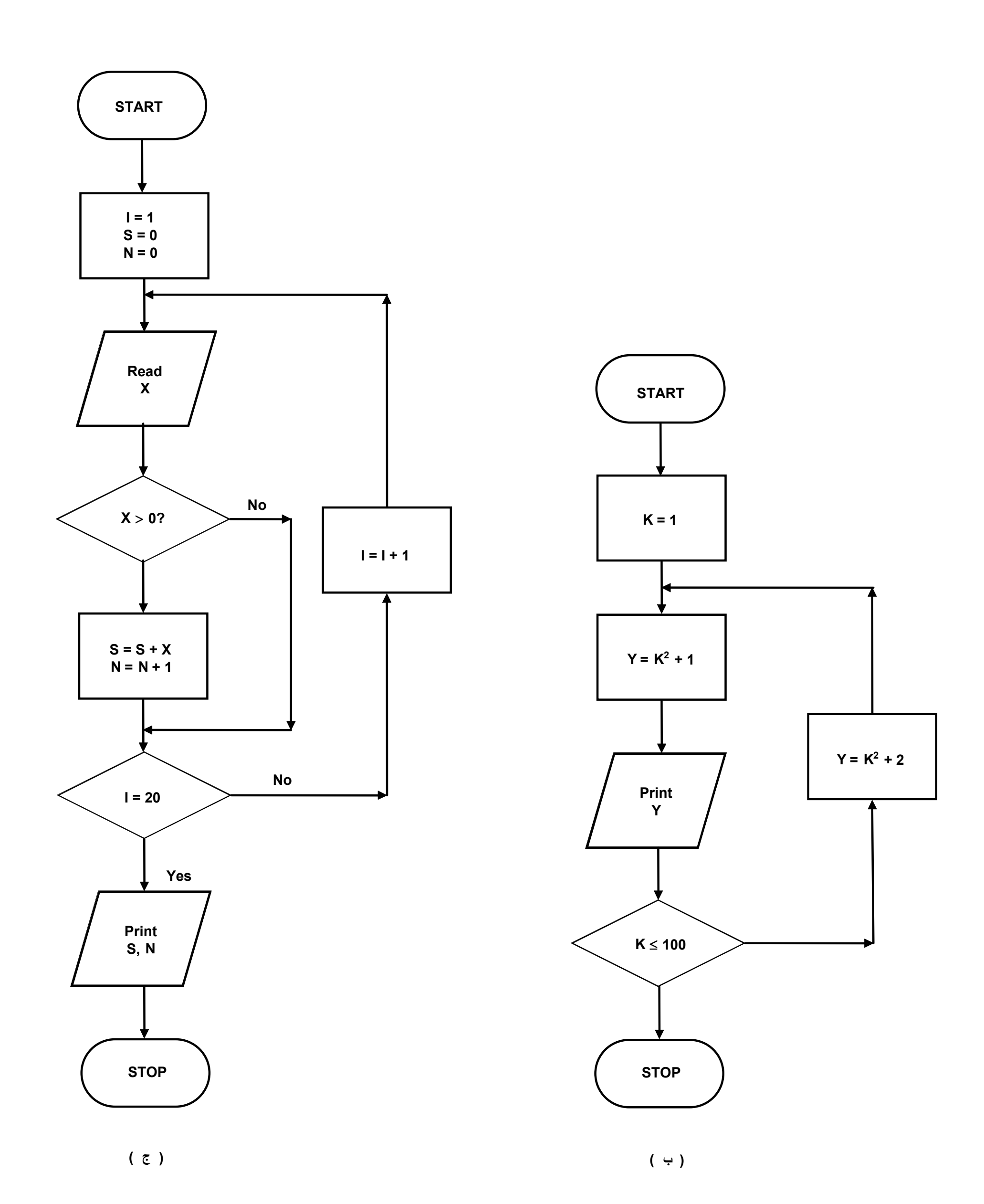
شكل (1-23)

تدريبات

(١) ادرس المخططات (أ،ب،ج) مبيناً أهداف كل مخطط والنواتج النهائية التي سيطبعها الحاسب عند تنفيذ التعليمات المبينة إزاء كل مخطط ، x=3 , Y=5 .



(أ)


START
I = 1
S = 0
N = 0
Read
X
X > 0?
No
S = S + X
N = N + 1
I = I + 1
I = 20
No
Yes
Print
S, N
STOP
START
K = 1
Y = K2 + 1
Print
Y
K ≤ 100
Y = K2 + 2
STOP

(ج)

(ب)

٢- ارسم خريطة سير البرنامج التي تمثل كلا من الخوارزميات التالية:

(أ)

1- ضع قيمة SUM صفراً، وقيمة N تساوي 1

2- اجمع N إلى SUM

3- إذا كانت N<6 فأضف 1 إلى قيمة N الحالية، ثم اذهب إلى الخطوة 1

4- اطبع قيمة SUM

(ب)

1- اقرأ قيمة X

2- إذا كانت $X \geq 0$ فاذهب إلى الخطوة (5)

3- احسب قيمة W من المعادلة $W = \sqrt{x^2 + 5x - 4}$ ، ثم اذهب إلى الخطوة (5)

4- احسب قيمة W من المعادلة: $W = -X + 13$

5- اطبع قيمتي X و W

3- ارسم خريطة سير البرنامج لحساب كلٍ من الاقترانات الآتية:

(أ) $f(X) = | X-3 |$

(ب) $SUM = \sum_{i=1}^{n} i$

(ج) $F = n! = n(n-1) \ldots (2)(1)$

(د) إيجاد قيمة أكبر عدد في المجموعة S حيث:

$S = [A, B, C]$

(هـ) إيجاد قيمة أصغر عدد في المجموعة S نفسها تنازلياً ثم تصاعدياً.

(ز) إيجاد قيمة أكبر عدد في السلسلة الحسابية:

$a_1, a_2, a_3, \ldots.., a_{n-1}, a_n$

(ح) ارسم خريطة سير البرنامج لإيجاد قيمة أصغر عدد في المتسلسلة الحسابية في السؤال السابق.

(ط) ارسم خريطة سير البرنامج بحيث ترتب حدود المجموعة التالية ترتيباً تنازلياً:

$a_1b_1, a_2b_2, a_3b_3, \ldots.., a_{n-1}b_{n-1}, a_nb_n$

(ى) ارسم خريطة سير عمليات لترتيب حدود المجموعة ترتيباً تصاعدياً.

(ل) اكتب أول 200 حد في المتوالية الهندسية التي تبدأ بالعدد 5، بحيث يكون معدل التغير 3.

(م) اكتب أول ثلاثين حداً في المتسلسلة التالية:

1, 3, 5, 7, ….

(ن) أوجد قيمة الاقتران عديد الحدود poly: حيث:

$$POLY = 1+Z+Z^2+ \ldots +Z^{10}$$

إذا كانت قيمة المتغير z معروفة لديك.

يعبأ هذا النموذج عن طريق المدرب

اسم المتدرب : - - - - - - - - **التاريخ** - - - - - -

رقم المتدرب : - - - - - - - - - - **المحاولة ١ ٢ ٣ ٤**

كل بند أو مفردة يقيم بـ ١٠ نقاط

العلامة : الحد الأدنى : ما يعادل ٨٠ ٪ من مجموع النقاط

الحد الأعلى : ما يعادل ١٠٠٪ من مجموع النقاط

بنود التقييم	النقاط
• تحديد أجزاء المشكلة الرئيسة • تحديد أجزاء المشكلة الفرعية • تقسيم المشكلة إلى أجزاء صغيرة • تحديد احتياجات حل المشكلة • معرفة رموز رسم خرائط التدفق • رسم خرائط التدفق للمشاكل البسيطة • رسم خرائط التدفق للمشاكل المتوسطة • وضع خوارزمية الحل للمشاكل المتوسطة	

ملاحظات: -
- -

توقيع المدرب: -

يعبأ هذا النموذج عن طريق المتدرب

<table>
<tr><td colspan="5">تعليمات
بعد الانتهاء من التدريب على **حل المشكلة** قيِّم نفسك بواسطة إكمال هذا التقييم الذاتي لكل عنصر من العناصر المذكورة، وفي حالة عدم قابلية المهمة للتطبيق ضع العلامة في الخانة الخاصة بذلك.</td></tr>
<tr><td colspan="5">اسم النشاط التدريبي الذي تم التدرب عليه: **حل المشكلة**</td></tr>
<tr><td rowspan="2">العناصر</td><td colspan="4">مستوى الأداء (هل أتقنت الأداء)</td></tr>
<tr><td>غير قابل للتطبيق</td><td>لا</td><td>جزئيا</td><td>كليا</td></tr>
<tr><td>• تحديد أجزاء المشكلة الرئيسة
• تحديد أجزاء المشكلة الفرعية
• تقسيم المشكلة إلى أجزاء صغيرة
• تحديد احتياجات حل المشكلة
• معرفة رموز رسم خرائط سير البرنامج
• رسم خرائط التدفق للمشاكل البسيطة
• رسم خرائط التدفق للمشاكل المتوسطة
• وضع خوارزمية الحل للمشاكل المتوسطة والبسيطة</td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5">يجب أن تصل النتيجة لجميع المفردات (البنود) المذكورة إلى درجة الإتقان الكلي أو أنها غير قابلة للتطبيق، وفي حالة وجود مفردة في القائمة "لا" أو جزئيا فيجب إعادة التدرُّب على هذا النشاط مرة أخرى بمساعدة المدرب</td></tr>
</table>



برمجة الحاسب

مكونات لغة الجافا

مكونات لغة الجافا

٢

الجدارة:

أن يكون المتدرب قادرا على كتابة الشفرة البرمجية code للبرامج البسيطة نسبيا

الأهداف :

عندما تكمل هذه الوحدة يكون لديك القدرة على:

١ - فهم ومعرفة واستخدام المتغيرات والثوابت
٢ - معرفة وكتابة جمل التعليق المختلفة
٣ - كتابة برامج بسيطة بلغة الجافا
٤ - استخدام جمل الإسناد
٥ - فهم واستعمال الأنواع المختلفة للبيانات مثل الأعداد الصحيحة والأعداد العشرية بأحجامها المختلفة بالإضافة إلى النصوص والأحرف والأعداد المنطقية
٦ - معرفة واستخدام العوامل المختلفة وكتابة التعبيرات البرمجية بلغة الجافا
٧ - كتابة التعبيرات الحسابية والمنطقية المختلفة
٨ - معرفة واستخدام أولويات تنفيذ العمليات المختلفة
٩ - تحويل الأعداد من نوع إلى نوع آخر
١٠ -استخدام الأقواس

مستوي الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة 100 ٪

الوقت المتوقع للتدريب : ٢٤ ساعة

الوسائل المساعدة:

- حاسب إلى
- قلم
- دفتر

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة

مكونات لغة الجافا

Components of Java programming languages

هذه الوحدة تبحث في أبجديات مكونات لغة الجافا (java) والتي تتكون من المتغيرات (Variables) والثوابت (Constants) والكلمات المحجوزة (Reserved Words) وغيرها والتي سوف نعرضها في هذه الوحدة . وسوف نتعرض بالشرح أيضا لجميع العمليات الحسابية(arithmetic operation) والمنطقية (logic) وعمليات الإسناد(assignment) والعمليات العلاقية (Relational operation) والنصية (String) وغيرها من العمليات.

أولا : تمثيل البيانات الأولية

في هذا الجزء من هذه الوحدة سوف نتعرف على بعض العناصر الأساسية والتي تستخدم في بناء برامج الجافا مثل المتغيرات والثوابت وغيرها. وسوف نقوم بشرح هذه المكونات من خلال أمثلة مكتوبة بلغة الجافا. وقبل الحديث عن مكونات لغة الجافا يجب أن نعلم أن البرامج المكتوبة بلغة الجافا تنقسم إلى نوعين : -

النوع الأول فيها يسمى برامج التطبيقات **(Application Program)** وهي برامج مكتوبة بلغة الجافا ويمكن تنفيذها مباشرةً من خلال بيئة الجافا باستخدام مفسر الجافا"Java Interpreter" .

أما النوع الثاني فهو ما يسمى بـ **Applet Program** وهذه البرامج يتم تنفيذها من خلال متصفحات الانترنت مثل Internet Explorer أو Netscape Navigator أو غيرها من متصفحات الانترنت وبالتالي يمكن تنفيذ هذه البرامج على أي حاسب ومع أي متصفح للانترنت وهذا ما يؤكد خاصية الحمل (النقل) لبرامج الجافا أي إمكانية تنفيذها على حاسب يدعم متصفحات الانترنت.

وسوف نستعرض في هذا الفصل برامج التطبيقات أما بالنسبة لبرامج Applet كيف يمكن كتابتها وترجمتها وتنفيذها سواء من خلال بيئة العمل أو من خلال المتصفحات؟، انظر ملحق "ب".

وكما ذكرنا سابقاً أننا سوف نتعرف على مكونات الجافا من خلال أمثلة وذلك بشرح هذه الأمثلة والتعليق عليها سطرا سطرا.

مثال١

اكتب برنامجاً تطبيقياً بسيطاً بلغة الجافا يطبع العبارة التالية

Welcome to Java Programming!

الحل :

البرنامج مبين في شكل (1-2)

```
1.   // Fig. 2-1: Welcome1.java
2.   // A first program in Java.
3.
4.      public class Welcome1 {
5.   // main method begins execution of Java application
6.
7.    public static void main( String args [  ] )
8.   {
9.   System.out.println( "Welcome to Java Programming!" );
10.
11.  }  // end method main
12.
13.  }  // end class Welcome1
```

Welcome to Java Programming! خرج البرنامج

شكل رقم (1-2) البرنامج الأول والخرج الناتج منه

البرنامج المبين في شكل (1-2) برنامج يطبع عبارة الترحيب Welcome to Java Programming!

ومن دراسة هذا البرنامج يتضح الآتي: -

1- إن بعض الحروف كتبت صغيرة small والبعض الآخر كتبت كبيرة capital وهذا يعني أن الحروف الكبيرة تختلف عن الحروف الصغيرة بالنسبة للمترجم، ولذلك ينبغي أخذ الحيطة والحذر الشديد عند كتابة البرنامج والتقيد بالكتابة بالحروف الكبيرة أو الصغيرة عند استخدام أسماء المتغيرات وغيرها ، فمثلاً الحاسب يفرق بين كل من الاسمين التاليين وهما مختلفان sum, Sum لأن أحدهما يبدأ بحرف كبير والآخر يبدأ بحرف صغير ولذلك فإن المترجم يعاملهما مختلفين. فالجافا تعتبر من اللغات الحساسة لحالة الحرف أي لا تتساوى فيها الحروف الكبيرة Capital Letters والحروف الصغيرة Small Letters .

2- إن كل البرامج التي ستذكر في هذه الحقيبة سوف يتم وضع رقم للسطر حتى وإن كان خالياً لا يحتوي على شيء وذلك لسهولة التعليق عليها وسهولة الإشارة إليها ويجب أن نعلم أن هذه الأرقام ليست جزءاً من برنامج الجافا ولا يجب كتابتها عند كتابة البرنامج.

شرح البرنامج

السطر الأول **// Fig. 2-1: Welcome1.java**

جملة من جمل التعليق.

جملة التعليق Comment Statement

تبدأ بـ// ثم يأتي بعدها أي نص مثل سطر١ ، سطر٢ ، وجمل التعليق يتم إهمالها أثناء ترجمة البرنامج وتنفيذه فهي جملة غير تنفيذية.

وتستخدم جمل التعليق لشرح البرنامج وتوثيقة داخلياً وكذلك للتعريف بوظيفة كل جزء وهي تسهل قراءة البرنامج وتعطي فكرة عن وظيفة كل جزء فيه عند كتابتها. وجمل التعليق قد تأتي في سطر واحد فقط أو جزء من سطر وفي هذه الحالة يجب أن تسبق بـ // أما إذا زادت جملة التعليق عن سطر فإنه في هذه الحالة يتم استخدام * / delimiter بحيث تبدأ بها الجمل وتنتهي بـ * / delimiter.

مثال على ذلك

```
/* This is a multiplier line
comment it can be split
into several lines */
```

وكل الجمل بين /*........... */ يتم إهمالها بواسطة المترجم Compiler، وجمل التعليق تفيد المبرمج في أنها تتيح له الفرصة لإضافة أي شرح لأي جزء من أجزاء البرنامج، ويمكن كتابة جمل التعليق بين العلامتين و / * و **/ وفي هذه الحالة يمكن استخدام خاصية من خصائص البرمجة بلغة الجافا وهي Javadoc لكي تقوم بقراءة البرنامج وتجميع كل التعليقات الموجودة فيه لعمل توثيق كامل للبرنامج ولكننا لن نتعرض لهذه الخاصية لأنها خارج نطاق هذه الحقيبة.

السطر الثاني **// A first program in Java**

جملة تعليق ثانيه تبين الغرض من البرنامج .

السطر الثالث سطر فارغ – المبرمج يستخدم الأسطر الفارغة والفراغات البينية لكي يُسهل قراءة البرنامج، والأسطر الفارغة والمسافات الفارغة تُهمل بواسطة المترجم ويمكن استخدامها وقتما يشاء المبرمج.

السطر الرابع

public class Welcome1 {

وهو يبدأ بتعريف الكائن class وإعطائه اسم (identifier). كل برنامج بلغة جافا يحتوي على الأقل على تعريف لكائن واحد يقوم المبرمج بتعريفه. وهذه الكائنات هي الكائنات المعرفة عن طريق المستخدم User defined classes.

والكلمة class تقوم بتعريف الكائن ويتبعها اسم هذا الكائن وهو Welcome1 (في هذا البرنامج).
والكلمة class من الكلمات المحجوزة في اللغة التي لها استخدامات خاصة ولذلك لا تصلح لأن تستخدم كاسم معرفي (identifier).

وعند كتابة أسماء الكائنات يفضل أن يبدأ الحرف الأول في اسم class بحرف كبير مثل Welcome1. .وكذلك إذا كان يتكون من أكثر من اسم فإن كل اسم يبدأ بحرف كبير مثل SampleClassName واسم الكائن يعرف بالاسم المعرفي identifier .

الاسم المعرفي identifier

يتكون الاسم المعرفي من مجموعة من الحروف (a-z, A-Z) والأرقام (9 → 0) بالإضافة إلى _ ، $
ويجب أن يراعى عند اختيار الاسم ما يلي : -

١ - أن يبدأ الاسم بحرف.
٢ - أن لا يبدأ برقم .
٣ - لا يحتوي على مسافة فارغة.
٤ - لا يكون من الأسماء المحجوزة (راجع قائمة الأسماء المحجوزة بشكل(2-2)).
٥ - يفضل أن يكون اسماً معبراً عن ما يقوم به الكائن.
٦ - لا يحتوي على أي حروف أو علامات خاصة أخرى غير المذكورة سابقاً.

ومن الأمثلة على ذلك

Welcome1, $Value, _Value.......S_ identified, ... etc

ومن الأمثلة الخاطئة للاسم المعرفي ما يلي: -

a) 7button لأنه يبدأ برقم.
b) Input filed1 لأنه يحتوي على مسافة.
c) Sum+total يحتوي على "+".
d) public كلمة محجوزة

Java Keywords الكلمات المحجوزة في لغة الجافا		
abstract	finally	public
boolean	float	return
break	for	short
byte	if	static
case	implements	super
catch	import	switch
char	instanceof	synchronized
class	int	this
continue	interface	throw
default	long	throws
do	native	transient
double	new	true
else	null	try
extends	package	void
false	private	volatile
final	protected	while

شكل (2-2) الكلمات المحجوزة في لغة الجافا

ونذكر مرة أخرى بأن الحروف الكبيرة لا تساوي الحروف الصغيرة في لغة الجافا. وخلال هذه الحقيبة عند تعريف الكائن (class) يجب أن يبدأ بكلمة public. وعند حفظ البرنامج في ملف يجب أن يكون اسم الملف هو نفسه اسم الكائن class متبوعاً بـ ".java" وكل الكائنات المعرفة عن طريق المستخدم يجب أن تحفظ في ملفات لها الامتداد ".java" وسوف يعطي المترجم خطأ عند الترجمة إذا لم يكن اسم الملف هو نفس اسم الكائن. وكذلك إذا لم يكن امتداد الملف java.

والقوس الأيسر في نهاية السطر الرابع } يبين بداية تعريف الكائن (class) ويجب أن ينتهي الكائن (class) بالقوس الأيمن { كما في السطر الثالث عشر من البرنامج.

خطأ شائع:

١. حفظ البرنامج في ملف باسم مختلف عن اسم الكائن (class) حتى ولو كانت نفس الحروف ولكنها تختلف عنها في الحروف الصغيرة والكبيرة يعطي عبارة خطأ عند الترجمة.
٢. استخدام امتداد للملف غير الامتداد المطلوب وهو java. يعطي أيضاً عبارة خطأ عند الترجمة.

ملحوظة : الأسطر التي تمثل جسم الكائن يفضل إزاحتها إلى اليمين قليلاً وذلك لتسهيل القراءة وتسهيل متابعة البرنامج وهذه الإزاحة تهمل عند الترجمة بواسطة المترجم أو المفسر .

السطر الخامس

// main method begins execution of Java application

هو جملة تعليق تبين الغرض من الأسطر 6-11 من البرنامج في شكل (١)

السطر السادس سطر فارغ لتسهيل القراءة

ملاحظة : يمكن إضافة سطر فارغ من الكتابة في أي مكان وذلك لتيسير القراءة

السطر السابع

public static void main(String args[ ])

يمثل جزءاً من كل تطبيق جافا (Java Application) حيث يبدأ تنفيذ البرنامج من الـ main ، والأقواس بعد الـ main توضح أن الـ main هو أحد المقاطع الرئيسة (block) في بناء التطبيق ويسمى method(الطريقة). وكل كائن (class) يجب أن يحتوي على الأقل على طريقة (method) واحدة وقد يحتوي على أكثر من طريقة ويجب أن تكون واحدة من هذه الطرق على الأقل تسمى main ويجب أن تعرف كما في السطر السابع. وفي حالة عدم وجود الmain فإنه لن يتم تنفيذ أي جزء من أجزاء البرنامج. والطرق (methods) تقوم بمعالجة البيانات وأداء بعض العمليات وبالتالي ينتج عنها بعض البيانات أو الخرج عند اكتمال تنفيذها.

والكلمة المحجوزة void تبين أن (الطريقة) method سوف تقوم بأداء عملية ما مثل (طباعة سطر - حساب مضروب عدد ما – حساب المتوسط الحسابي .... الخ)

في هـذه الحقيبـة سيكـون السـطر السـابع هـو أول سـطر في بدايـة method main وسـوف يأخـذ نفس الشكل الموجود في السطر السابع.

السطر الثامن القوس الأيسر} يحدد بداية method main.

بينما السطر الحادي عشر القوس الأيمن { يحدد نهاية method main .

وكما تم إزاحة الأسطر التي تُكَوِّن الـ class فإنه لتسهيل القراءة والبرمجة فكذلك يمكن إزاحة الأسطر التي تمثل جسم الـ main إلى اليمين قليلاً لنفس السبب.

السطر التاسع

System.out.println("Welcome to Java Programming!");

يخبر الكمبيوتر بطباعة الجملة !Welcome to Java Programming والموجودة بين علامات التنصيص " ". والجملة بين علامات التنصيص تسمى String والمسافات الفارغة في String تهمل بواسطة المترجم .

الجملة System.out تعرف بأنها جملة الخرج القياسية Standard Output Object، .وهذه الجملة تقوم بإظهار الجمل النصية وكذلك أي معلومات أو بيانات في نافذة الأوامر حيث يتم تنفيذ برامج الجافا. والـ Method المسماة System.out.println في هذا البرنامج تظهر النص في سطر واحد في نافذة الأوامر Command Window وعندما تنتهي الطباعة فإن المؤشر يوضع في بداية السطر التالي، وهذا يماثل ضغط مفتاح Enter في لوحة المفاتيح عند الكتابة.

وفي نهاية السطر وضعت الفاصلة المنقوطة ; وهذا يعني أن جملة جافا (Java Statement) قد انتهت. وكل جملة من جمل الجافا يجب أن تنتهي بفاصلة منقوطة. والفاصلة المنقوطة تحدد نهاية الجملةStatement Terminal.

خطأ شائع:

عدم وضع فاصلة منقوطة في نهاية جملة الجافا (أحياناً تكتب الجملة على أكثر من سطر) فإن المترجم يعتبر الجملة غير منتهية ويعطي خطأ.

بعض المبرمجين يجد صعوبة أو عدم وضوح عند استخدام الأقواس ولذلك فإنهم يفضلون كتابة جملة تعليق توضح هل القوس نهاية class أو method أو غيرهما. كما هو واضح في السطر الحادي عشر الذي ينهي method ولذلك تستخدم جملة تعليق مثل الموجودة في السطر الحادي عشر والثالث عشر.

السطر الحادي عشر **} // end method main**

والسطر الثالث عشر (class) **} // end class Welcome1 .**

ترجمة وتنفيذ البرنامج الأول

نحن الآن نستطيع ترجمة وتنفيذ هذا البرنامج. ولترجمة البرنامج فإننا نستطيع عمل ذلك من خلال كتابة الأوامر في نافذة الأوامر أو باستخدام القوائم الموجودة في بيئة التشغيل المستخدمة مع Java مثلForte ,Kawa أو غيرهما . وسوف نستعرض أولاً الترجمة والتنفيذ من خلال نافذة الأوامر ثم بعد ذلك نشرح كيف تتم الترجمة والتنفيذ من خلال بيئة العمل .

أولاً الترجمة والتنفيذ باستخدام نافذة الأوامر

١ - غير الفهرس إلى الفهرس الذي تم حفظ البرنامج فيه ثم اكتب الأمر التالي
Javac Welcome1.Java

إذا كان البرنامج يحتوي على أخطاء بنائية (Syntax Errors) فإن هذه الأخطاء سوف تظهر في نافذة الأوامر موضحاً فيها رقم السطر ومكان الخطأ وتفسير محتمل للخطأ. وفي هذه الحالة يجب تصحيح هذه الأخطاء في البرنامج ثم إعادة هذه الخطوة السابقة ثانية ويتم تكرارها حتى يصبح البرنامج بدون أخطاء ويعطي العبارة التالية No Errors . وفي هذه الحالة فإن المفسر يقوم بإنشاء وحفظ ملف جديد يسمى Welcome1.class يحتوي على الـ Byte code. هذا الملف ينتج من ترجمة جمل لغة الجافا بواسطة المترجم إلى byte code وهي صورة أخرى للبرنامج وهي الصورة التنفيذية للبرنامج. و لتنفيذ البرنامج من خلال نافذة الأوامر نكتب الأمرjava Welcome1 في نافذة الأوامر. وإذا لم يتوفر الملف ذو الامتداد class. فإن المفسر لا يستطيع تنفيذ البرنامج ويعطي رسالة خطأ.

وتنفيذ البرنامج يبدأ من main method ثم ينتقل إلى الجمل التنفيذية فيه والتي تقوم بإظهار الجملة بين علامتي التنصيص. وعند تنفيذ البرنامج فإن المفسر يقوم بتنفيذ Byte Code الناتج من عملية الترجمة والموجود في الملف ذي الامتداد class.

ثانيا: تنفيذ البرنامج من خلال بيئة العمل

يتم كتابة وترجمة وتنفيذ البرنامج من خلال بيئة العمل Forte ، انظر ملحق "أ".

تعديل البرنامج الأول

في هذا الجزء سوف يتم شرح مثالين يعتمدان على المثال الأول . الأول منهما يقوم بطباعة النص السابق في سطر واحد باستخدام جملتين، أما الثاني فإنه يقوم بطباعة النص على أكثر من سطر باستخدام جملة Println واحدة.

إظهار سطر نصي باستخدام أكثر من جملة

الجملة Welcome to Java Programming! سوف يتم إظهارها في سطر واحد باستخدام جملتين، والبرنامج الذي يقوم بذلك مبين في شكل(٤).

مثال٢ : برنامج يظهر النص **Welcome to Java Programming!** في سطر واحد باستخدام أكثر من جملة طباعة

```
1. // Fig. 2-3: Welcome2.java
2. // Printing a of text line with multiple statements.
3.
4. public class Welcome2 {
5.
6. // main method begins execution of Java application
7.  public static void main( String args[ ] )
8. {
9.         System.out.print( "Welcome to " );
10.    System.out.println( "Java Programming!" );
11.    }  // end method main
12.    }  // end class Welcome2
```

شكل (2-3) برنامج إظهار سطر باستخدام أكثر من جملة

ومعظم جمل هذا البرنامج تشابه البرنامج المبين بشكل (2-1) ولذلك سوف نعلق فقط على الأسطر الجديدة غير المتشابهة.

السطر الثاني **// Printing a of text line with multiple statements.**

جملة تعليق تبين الهدف من البرنامج

السطر الرابع **public class Welcome2 {**

تعريف الكائن وإعطاؤه اسم Welcome2.

السطر التاسع والسطر العاشر من الـ **method main** .

```
System.out.print( "Welcome to " );
System.out.println( "Java Programming!" );
```

تظهران سطراً واحداً من النصوص في نافذة الأوامر . الجملة الأولى تظهر النص Welcome to ثم تضع المؤشر في نهاية هذا السطر بينما الجملة الثانية تبدأ من نهاية هذا السطر وتظهر Java Programming! بعد كلمة to وبعد الطباعة تضع المؤشر في بداية السطر التالي.

الفرق بين println, print يظهر بعد كتابة ما بين القوسين ففي حالة print يتم وضع المؤشر في نهاية الجملة التي تمت كتابتها ، أما في حالة println فإنه يتم وضع المؤشر في بداية السطر التالي. ولذلك فإن السطر العاشر سوف يظهر بعد آخر حرف في الجملة الموجودة في السطر التاسع حيث وضع المؤشر بعد انتهاء تنفيذ هذه الجملة.

إظهار عدد من الأسطر باستخدام جملة واحدة

إظهار نص من النصوص في أكثر من سطر واحد باستخدام جملة واحدة يتم ذلك باستخدام ما يسمى حرف السطر الجديد"\n " NewLine Character.

```
1.    // Fig. 2-4: Welcome3.java
2.    // Printing multiple lines with a single statement.
3.
4. public class Welcome3 {
5.
6.    // main method begins execution of Java application
7.    public static void main( String args[  ] )
8.
9.    {
10.   System.out.println( "Welcome\nto\nJava\nProgramming!" );
11.   }  // end method main
12.   }  // end class Welcome3
```

خرج البرنامج

```
Welcome
to
Java
Programming!
```

شكل (2-4) برنامج طباعة أكثر من سطر باستخدام جملة واحدة وخرجه

وشكل (2-4) يبين البرنامج وكذلك الخرج الناتج من البرنامج حيث يطبع الجملة في أربعة أسطر باستخدام الحرف الخاص بالسطر الجديد" n \"حيث يوضع في المكان المراد بدأ سطر جديد فيه وسوف نشرح الجمل المختلفة عن البرنامج السابق.

مثال ٣ : برنامج يظهر العبارة **Welcome t o Java Programming!** في أربعة أسطر كل كلمة في سطر

// Printing multiple lines with a single statement.

السطر الثاني

تعليق يوضح الهدف من البرنامج وهو طباعة العديد من الأسطر باستخدام جملة واحدة.

السطر الرابع

public class Welcome3 {

تعريف الكائن وإعطاؤه اسم Welcome3 .

السطر العاشر

System.out.println("Welcome\nto\nJava\nProgramming!);

هذه الجملة تظهر أربعة أسطر من النص في نافذة الأوامر. الأحرف المكونة للنص الموضوعة بين علامتي التنصيص يتم إظهارها كما هي بالضبط ولكن الحرفان \ ,n لم يتم طباعتها على الشاشة. والشرطة الخلفية (\ ) تسمى حرف escape وتعتبر من الحروف الخاصة التي تستخدم في جمل الخرج . عند استخدام الشرطة الخلفية في داخل نص فإن الجافا تقوم بضم الحرف التالي للشرطة الخلفية ليكونا معا ما يسمى escape sequence فمثلاً الـ escape sequence \n يعرف بحرف السطر الجديد الذي يحرك المؤشر إلى بداية السطر التالي في نافذة الأوامر.

وشكل (2-5) يوضح بعض الـ escape sequence المعروفة.

الحرف الخاص	الوصف
\n	سطر جديد. يضع المؤشر في بداية السطر التالي
\t	مسافة أفقية. تحريك المؤشر مسافة معينة إلى النقطة التالية في السطر
\r	carriage return. يضع المؤشر في بداية السطر الحالي ولا يتقدم إلى السطر التالي ، وأي حرف يطبع يتم طباعته على حرف سابق تم كتابته في نفس السطر
\\	شرطة خلفية. إظهار "\ " في الخرج
\"	علامة تنصيص مزدوجة. إظهار علامة التنصيص المزدوجة

شكل (2-5) escape sequence

إظهار نص في صندوق الحوار

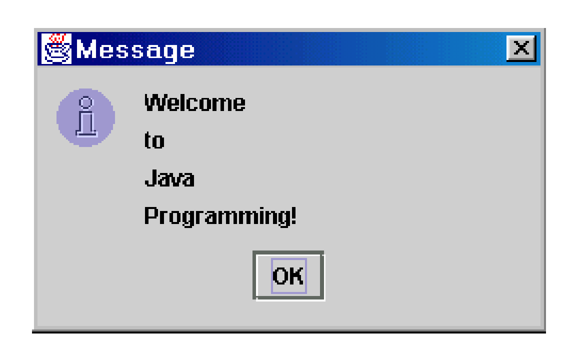
بالرغم من إظهار النص السابق في نافذة الأوامر، إلا أن الكثير من تطبيقات الجافا تستخدم صناديق الحوار لإظهار النصوص بدلاً من نافذة الأوامر، معظم البرامج وبخاصة متصفحات الانترنت مثل Microsoft internet explorer, Netscape Navigator تستخدم صناديق الحوار في كثير من التطبيقات.

وصناديق الحوار هي عبارة عن نافذة يتم إظهار الرسائل المهمة الموجهة للمستخدم فيها، أو التي تعطي خرجاً من البرنامج والـ class المسمى JOptionPane يمدنا بالـ methods التي تساعدنا في إظهار صناديق الحوار المختلفة.

مثال٤

طباعة النص **Welcome to Java Programming!** باستخدام صندوق حوار

```
1. // Fig. 2-6 : Welcome4.java
2. // Printing multiple lines in a dialog box
3.
4. // Java extension packages
5. import javax.swing.JOptionPane;  // import class JOptionPane
6. public class Welcome4 {
7.
8. // main method begins execution of Java application
9. public static void main( String args[  ] )
10.        {
11.        JOptionPane.showMessageDialog(
12.        null, "Welcome\nto\nJava\nProgramming!" );
13.
14.        System.exit( 0 );  // terminate application
15.        }  // end method main
16.        }  // end class Welcome4
```



شكل(2-6) كود البرنامج وصندوق الحوار لإظهار نص في أكثر من سطر

وشكل(2-6) يظهر نفس النص السابق في صندوق حوار يسمى message Dialog. وأحد عناصر القوة في لغة الجافا هو احتواؤها على العديد من الكائنات الجاهزة التي يمكن للمبرمجين إعادة استخدامها ثانيةً بدلاً من إنشائها من البداية.وهذه الكائنات الجاهزة والمحددة مسبقاً يتم تنظيمها وتجميع المتعلق بعضه ببعض في حزم (packages) وهذه الحزم عبارة عن مجموعة من الكائنات (classes) التي تكوِّن مكتبة الجافا أو أنها تعرف بما يسمى بـ
java Application Programming Interface (java API). والجافا API تنقسم إلى قسمين أساسيين هما الحزم الأساسية (Core packages) وحزم الامتداد (Extension packages). اسم المجموعة الأساسية يبدأ بـ java بينما الامتداد يبدأ بـ javax. و كثير من هذه الحزم الآن تم دمجها في برنامج الجافا بدءاً من الإصدار الثاني. ويجدر الإشارة هنا إلى أنه نتيجة التطوير المستمر للغة الجافا فإنه يضاف إليها حزم جديدة يمكن تعريفها وتحميلها من الموقع Java.sun.com .
في البرنامج الموضح بشكل (2-6) تم استخدام الكائن JOptionPane الذي تم تعريفه ووضعه في الحزمة javax.swing

السطر الرابع **// Java extension packages**

جملة تعليق في سطر واحد تبين الجزء من البرنامج الذي يشير إلى استدعاء كائن معين من حزمة الامتداد باستخدام import.

وسوف تنقسم جمل import إلى مجموعات: -

١. جمل import للحزم الأساسية (Java).
٢. جمل import لحزم الامتداد (Javax).
٣. جمل import للحزم الخاصة بـDeitel وسوف يتم تعريفها في حينها.

السطر الخامس

import javax.swing.JOptionPane; // import class JOptionPane

يوضح جملة import ، والمترجم يستخدم جملة import لكي يتم تعريف وتحميل الكائنات المستخدمة في لغة الجافا ، وفي هذه الحالة استدعاء وتضمين للكائن المسمى JOptionPane.

وعند استخدام الكائنات من API فإن المترجم يتأكد من استخدامها بالصورة الصحيحة. وجملة import تساعد المترجم في أن يحدد ويجد الكائن ولذلك فإنه عند استخدام أي كائن من الجافا API يجب تعريف الحزمة التي تحتوي على هذا الكائن.

يمكنك معرفة بعض المعلومات عن الحزم و الكائنات الموجودة في الجافا API من الموقع Java . sun . com / j2se/1.3/docs/api/index.html كما يمكنك تنزيل الوثائق الخاصة بذلك على حسابك الشخصي من الموقع Java.sun.com / j2se/1.3/docs.html

خطأ شائع:

عدم الالتزام بنفس جملة import كما وردت في السطر الخامس من حيث الأحرف الكبيرة والصغيرة.

في السطر الخامس يخبر المترجم بأن يُحمل الكائن المسمى JOptionPane من الحزمة المسماة javax.swing . وهذه الحزمة تحتوى على كثير من الكائنات الأخرى مثل الكائنات الخاصة بالرسومات والتعامل مع المستخدم من خلال بيئة الرسومات GUI graphical user interface التي تسهل إدخال و إخراج البيانات من خلال مربعات حوار.

السطر الحادي عشر والثاني عشر

```
JOptionPane.showMessageDialog(
 null, "Welcome\nto\nJava\nProgramming!" );
```

يشير إلى استدعاء الـ method المسماة show.Message.Dialog من الكائن المسمى JOptionPane.وهذه الـ method تتطلب مدخلين (two arguments) مفصولين بفاصلة "،". المدخل الأول (first argument) دائماً سيكون الكلمة "null " ، وهو يحدد المكان الذي يظهر فيه صندوق الحوار ، وفي هذه الحالة (استخدام الكلمة "null ") فإن صندوق الحوار سوف يظهر في منتصف الشاشة ، أما المدخل الثاني second argument فهو النص المراد إظهاره.

والـ method المسماة showMessage هي method خاصة من الـكائن المسمى JOptionPane تسمى Static Method. وهذه الـ method دائماً تستدعى باستخدام اسم الـكائن متبوعاً بنقطة يليها اسم الـ method كما هو واضح في الشكل التالي

class name . method name (arguments)

تنفيذ السطرين 11-12 سوف يظهر صندوق الحوار المبين بشكل (2-7).
العنوان الذي يظهر في صندوق الحوار يحتوي على الكلمة Message يبين أن هذا الصندوق يظهر رسالة إلى المستخدم. وصندوق الحوار يحتوي على زر OK يسمح للمستخدم بإخفاء هذا الصندوق بالضغط على الزر OK.



شكل (2-7) صندوق حوار اظهار رسالة

ويجب أن تتذكر أن أي جملة داخل الـ method يجب أن تنتهي بفاصلة منقوطة. لغة الجافا تسمح بتقسيم الجمل الكبيرة على أكثر من سطر وفي هذه الحالة فإن الفاصلة المنقوطة تأتي في نهاية الجملة وليس في نهاية كل سطر. بعض الجمل لا يمكن تقسيمها في أماكن معينة منها، فمثلاً لا يمكن تقسيم الجملة في وسط الأسماء المعرفية(Identifier) أو في وسط أي نص.

السطر الرابع عشر **System.exit(0); // terminate application**

وهذه الجملة تشير إلى استخدام ثابت للـ method المسماة exit والموجودة في الـكائن المسمى System وذلك لإنهاء التطبيق. ويجب استخدام هذه الجملة في كل التطبيقات التي تستخدم GUI وذلك لإنهاء التطبيق. ونلاحظ أنه في السطر الرابع عشر استخدمت نفس القاعدة السابقة في استدعاء الـ method من داخل الـكائن بكتابة اسم الـكائن ثم نقطة ثم يلي ذلك اسم الـ method . الأسماء المعرفية للـكائن تبدأ دائماً بحرف كبير Capital. الـكائن system لم يتم استيراده أو دمجه مع

البرنامج لأنه جزء من الحزمة Java.lang، والحزمة Java.lang هي الحزمة الوحيدة التي لا يتطلب استخدام أيٍ من الـ methods المدمجة فيهاويتم استدعاؤها عن طريق جملةimport.
المدخل (argument) (0)للـ method exit يبين أن التطبيق تم إنهاؤه بنجاح وبدون أخطاء. وإذا كان المدخل لا يساوي الصفر فإن ذلك يعني وجود خطأ.

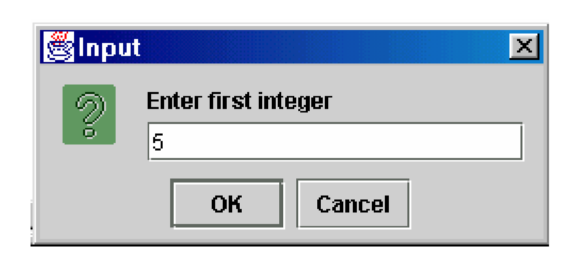
مثال٥

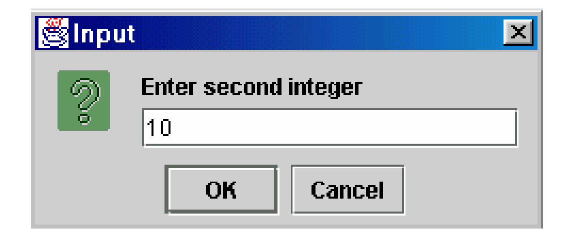
إضافة أرقام صحيحة

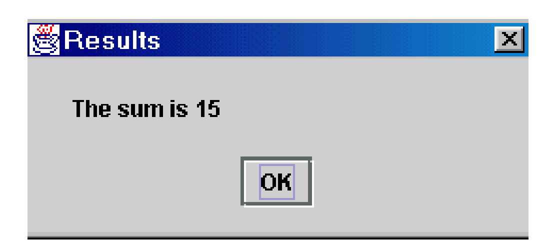
البرنامج التالي يقوم بقراءة رقمين صحيحين من خلال لوحة المفاتيح ثم يحسب مجموعهما ومن ثم يقوم بطباعة مجموع الرقمين
البرنامج وشاشات الإخراج والإدخال موضحة في شكل (2-8)

```
1. // Fig. 2-8 : Addition.java
2. // An addition program.
3.
4. // Java extension packages
5.  import javax.swing.JOptionPane;  // import class JOptionPane
6.
7. public class Addition {
8.
9. // main method begins execution of Java application
10.       public static void main( String args[ ] )
11.       {
12.       String firstNumber;   // first string entered by user
13.       String secondNumber;  // second string entered by user
14.       int number1;          // first number to add
15.       int number2;          // second number to add
16.       int sum;              // sum of number1 and number2
17.
18.       // read in first number from user as a string
19.       firstNumber =
20.       JOptionPane.showInputDialog( "Enter first integer" );
21.
```

```
22.        // read in second number from user as a string
23.        secondNumber =
24.        JOptionPane.showInputDialog( "Enter second integer" );
25.        // convert numbers from type String to type int
26.        number1 = Integer.parseInt( firstNumber );
27.        number2 = Integer.parseInt( secondNumber );
28.
29.        // add the numbers
30.        sum = number1 + number2;
31.
32.        // display the results
33.        JOptionPane.showMessageDialog(
34.        null, "The sum is " + sum, "Results",
35.        JOptionPane.PLAIN_MESSAGE );
36.
37.        System.exit( 0 );   // terminate application
38.
39.        }  // end method main
40.
41.        }  // end class Addition
```







شكل(8-2) برنامج إضافة رقمين مع شاشات إدخال وإخراج البيانات

شرح البرنامج

السطران الأول والثاني

```
// Fig. 2-8 : Addition.java
// An addition program.
```

تمثلان جملة تعليق توضح الشكل الذي يظهر فيه البرنامج وكذلك عمل البرنامج .

السطر الرابع يمثل جملة تعليق تبين أن السطر التالي يحتوي على جملة import التي تتضمن استدعاء (تضمين)الحزمة الممتدة للجافا مع البرنامج .

السطر الخامس

```
import javax.swing.JOptionPane;  // import class JOptionPane
```

تقوم هذه الجملة بإخبار المترجم بتحميل الكائن المسمى JOptionPane الموجود في الحزمة javax.swing للاستخدام في البرنامج. وكما ذكر من قبل فإن كل برنامج يجب أن يتكون من كائن واحد على الأقل كما هو مبين في

السطر السابع **public class Addition {**

ولذلك يجب أن يتم حفظ هذا التطبيق (البرنامج) في ملف له نفس اسم الكائن class وهو Addition ولذلك فإن اسم الملف يصبح Addition.java .

ويبدأ الكائن كما هو معروف بالقوس الأيسر } كما هو في السطر السابع وينتهي بالقوس الأيمن { كما في السطر ٤١. وكما هو معروف فإن كل تطبيق يبدأ التنفيذ بالـ method main (الأسطر من ١٠:٤٠) والتي تبدأ من السطر ١١ الذي يحتوي على القوس الأيسر وتنتهي عند السطر ٣٩ القوس الأيمن الذي يبين نهاية main.

السطر الثاني عشر والسطر الثالث عشر

```
String firstNumber;   // first string entered by user
 String secondNumber;  // second string entered by user
```

هذه الجمل تعرف بجمل التعريف declaration statements.

جمل التعريف declaration statements

هذه الجمل تقوم بتعريف المتغيرات Variables التي سوف تستخدم في داخل البرنامج بتحديد اسم لها وكذلك تحديد نوع البيانات التي ستخزن فيها، ويجب أن تأتي جمل التعريف في بداية الطريقة method لأن لغة الجافا لا تسمح باستخدام أي من المتغيرات إلا بعد تعريفها و أي متغير يستخدم خلال البرنامج يجب أن يكون له اسم وأسماء المتغيرات تتبع نفس القواعد التي ذكرت عند الحديث عن الأسماء المعرفية identifiers ونُذَّكر بهذه القواعد مرةً ثانية وهي: -

- يتكون الاسم من حروف الهجاء a-z ، A-Z والأرقام ($0 \rightarrow 9$) ، $ ، _
- أن لا يبدأ الاسم برقم
- لا يحتوي على مسافات فارغة
- لا يكون من الكلمات المحجوزة
- لا يحتوي على علامات خاصة غير $، _

والمتغير يمثل مكاناً في ذاكرة الجهاز حيث سيتم تخزين فيه المتغير في هذا المكان بواسطة البرنامج. ففي هذا البرنامج الكلمات firstNumber و secondNumber هي أسماء متغيرات من النوع String (الموجودة في الحزمة Java.lang) وهذا يعني أن هذه المتغيرات سوف تحمل بيانات من نوع String. كل جملة تعريف يجب أن تنتهي بفاصلة منقوطة " ; "، وقد يضاف في نهايتها (بعد الفاصلة المنقوطة) جملة تعليق لبيان وتوضيح ما يحمله هذا المتغير، وهذه عادة المبرمجين لتوضيح الغرض من هذه المتغيرات حتى يسهل فهم البرنامج لمن يقرؤه وكذلك تعتبر طريقة لتوثيق البرنامج.

يمكن استخدام جملة تعريف لكل متغير كما هو مبين في السطرين الثاني والثالث عشر، كما يمكننا استخدام جملة واحدة لتعريف المتغيرات من نفس النوع وفي هذه الحالة يتم فصل المتغيرات بفاصلة. فالسطران الثاني عشر والثالث عشر يمكن كتابتهما في جملة تعريف واحدة كما يلي

```
String firstNumber,   // first string entered by user
secondNumber;  // second string entered by user
```

الأسطر من ١٤ - ١٦

```
14        int number1;        // first number to add
15        int number2;        // second number to add
16        int sum;         // and sum of number1 numbe
```

هذه الأسطر تعرف المتغيرات number1, number2 and sum بأنها بيانات من النوع int وهذا يعني أن هذه المتغيرات سيوضع بها قيمة عددية صحيحة مثل(7 ,11 ,200 ....الخ). وشكل (2-9) يبين الأنواع الأساسية للبيانات Primitive data type في لغة الجافا والحجم الذي يشغله هذا النوع بالبت (bits). وهذه الأنواع تعتبر من الكلمات المحجوزة في اللغة وتكتب دائماً في بالحروف الصغيرة.

النوع (type)	الحجم بالبت (Size in bits)	المدى (range) أو القيمة (value)	ملاحظات
boolean	١	True or false	قيمة منطقية
char	16	to FFFF٠٠٠٠	متغير يحمل حرفاً واحداً فقط
byte	8	-128 to +127	
short	16	-32,768 to +32767	قيمة صحيحة في المدى الموضح قرين كل نوع
int	32	-2,147,483,648 to +2,147,483,647	
long	64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
float	32	-3.40292347 E+38 to + 3.40292347 E+38	قيمة تحتوي على نقطة عشرية في المدى الموضح قرين كل نوع
double	64	-1.79769313488231570 E+308 to +1.79769313488231570 E+308	

شكل (2-9) أنواع البيانات الأساسية في لغة الجافا

السطر الثامن عشر

```
18    // read in first number from user as a string
```

يبين جملة تعليق توضح أن الجملة التالية ستقوم بقراءة الرقم الأول المدخل بواسطة المستخدم.

السطر التاسع عشر والعشرون

```
19  firstNumber =
20  JOptionPane.showInputDialog( "Enter first integer" );
```

هذه الجملة تقوم بقراءة String يمثل الرقم الأول الذي يجب أن يضاف والـmethod JOptionPane.showInputDialog تظهر مربع حوار لإدخال الرقم كما هو مبين بشكل (2-10).



شكل رقم (2-10) مربع حوار لإدخال قيمة

المدخل (argument) إلى مربع الحوار showInputDialog هو عبارة عن جملة توضح للمستخدم ما يجب عليه فعله وهذه الرسالة تسمى prompt لأنها ترشد المستخدم لعمل فعل معين. ويقوم المستخدم بإدخال البيانات النصية ثم يضغط OK أو يضغط Enter وعندها يتم حفظ المدخلات كنص في المتغير المسمى firstNumber الذي تم تعريفه مسبقا على أنه متغير نصي.

السطر الثاني والعشرون

```
22     // read in second number from user as a string
```

جملة تعليق تبين الغرض من السطرين التاليين وهو إدخال القيمة الثانية التي ستضاف والتي يتم إدخالها كنص

السطران ٢٣ -٢٤

```
23  secondNumber =
24              JOptionPane.showInputDialog( "Enter second integer" );
```

يظهران صندوق حوار لإدخال القيمة الثانية والتي تدخل كنص ويتم حفظها في المتغير المسمى secondNumber

السطر ٢٥

```
// convert numbers from type String to type int
```

جملة تعليق من سطر واحد تبين وظيفة السطرين التاليين وهي تحويل الرقمين السابقين من الصورة النصية إلى الصورة الرقمية.

السطران ٢٦ -٢٧

```
26   number1 = Integer.parseInt( firstNumber );
27   number2 = Integer.parseInt( secondNumber )
```

هاتان الجملتان تقومان بتحويل القيمة النصية المدخلة بواسطة المستخدم والتي تم حفظها في المتغيرين firstNumber, secondNumber إلى قيمة صحيحة حتى نستطيع استخدامها في حساب المجموع.

وال method المسماة Integer.parseInt (هي method ثابتة من الكائن Integer) تحول المدخل النصي إلى قيمة صحيحة. والكائن Integer معرف في الحزمة java.lang وبالتالي لا يتطلب استخدامه استدعاءه بجملة import. وبعد التحويل يتم حفظ القيمة المحولة في المتغيرين number1, number2 على التوالي.

لتجويل متغير نصي إلى متغير من النوع Double فإننا نستخدم الجملة التالية:

variable1= Double.parseDouble (*variable2*)

حيث variable1, variable2 متغيران من النوع double , String على التوالي.

السطر ٢٩

// add the numbers

جملة تعليق من سطر واحد لبيان الغرض من السطر القادم

السطر ٣٠

٢٩ sum = number1 + number2;

هذا السطر يقوم بحساب مجموع المتغيرين number1, number2 وذلك باستخدام عامل الجمع "+" ويحفظ الناتج في المتغير sum باستخدام عامل الإسناد "=" . جميع العمليات الحسابية تتم باستخدام جمل الإسناد، في عمليات الجمع تضاف القيم المخزنة في المتغيرات بعضها إلى بعض. ففي هذا المثال تضاف القيمة الموجودة في المتغير number1 إلى القيمة الموجودة في المتغير number2 ويحفظ الناتج في المتغير sum. ويتضح أيضا من هذا المثال أن عامل الجمع (+) هو عامل ثنائي المدخلات : أي يحتاج إلى معاملين وهما كما في هذا المثال number1, number2 وسوف نتعرض بالشرح لجميع العمليات الحسابية والمنطقية وغيرها من العمليات بالتفصيل في الجزء المتبقي من هذه الوحدة بعد الانتهاء من شرح هذا المثال.

الأسطر ٣٤ -٣٦

```
٣٤        JOptionPane.showMessageDialog(
٣٥        null, "The sum is " + sum, "Results",
٣٦        JOptionPane.PLAIN_MESSAGE );
```

تمثل جملة الخرج والتي تستخدم مربع الحوار لإظهار ناتج عملية الجمع
وهذه النسخة لـ method **JOptionPane.showMessageDialog** تتطلب أربعة مدخلات (Four arguments) هي: -

١ - المدخل الأول null وهو يخبر الحاسب بوضع صندوق الحوار في وسط الشاشة.

٢ - المدخل الثاني هو الرسالة التي سوف تظهر في صندوق الحوار وفي هذا المثال هي "The sum is " + sum وهنا تم استخدام العامل " + " لإلحاق القيمة المُخزَّنة في المتغير sum بعد تحويلها إلى نص في نهاية النص "The sum is ".

ويجب التفريق بين استخدام العامل " + " كعامل إضافة للقيمة الحسابية وعامل إلحاق بين النصوص في جملة الخرج (مربع الحوار).

٣ - المدخل الثالث يمثل النص الذي سوف يظهر في سطر العنوان لمربع الحوار، في هذا المثال "Results" (راجع شكل (2-8) السابق.

٤ - المدخل الرابع JOptionPane.PLAIN_MESSAGE يُمثل الرمز الذي يبين نوع مربع الحوار.

ويوجد مجموعة من الرموز التي يمكن اظهارها في صندوق الحوار لتساعد المستخدم في معرفة نوع صندوق الحوار والرسالة التي تظهر فيه وهذه الرموز مبينة في شكل (2-11)

نوع رسالة صندوق الحوار	الرمز	الوصف
JOptionPane. ERROR_ MESSAGE		عرض صندوق حوار يبين رسالة خطأ للمستخدم
JOptionPane. INFORMATION_ MESSAGE		عرض صندوق حوار مع رسالة للمستخدم
JOptionPane. WARNING_ MESSAGE		رسالة تحذيرية للمستخدم
JOptionPane. QUESTION_ MESSAGE	?	سؤال للمستخدم يجب الإجابة عليه بنعم أو لا
JOptionPane. PLAIN_ MESSAGE	لا يوجد رمز	يظهر رسالة في الصندوق بدون رموز

شكل (2-11) الرموز التي تظهر مع صندوق الحوار

أنواع العمليات

تتعدد العمليات في لغات البرمجة والتي تتيح للمبرمج من تحقيق الهدف من البرنامج وفيما يلي سنعرض أنواع هذه العمليات:

العمليات الإسنادية Assignments

تستخدم هذه العملية لتخصيص قيمة ما في متغير وذلك بعد تعريفه ، ونستخدم العملية = للتعبير عن التخصيص في لغة الجافا :

أمثلة:

x = 1 ;

تم تخصيص 1 إلى المتغير x

radius = 1.0 ;

تم تخصيص القيمة 1.5 إلى المتغير x

a = ‘A’ ;

تم تخصيص الحرف ‘A’ إلى المتغير a

خطأ شائع : تخصيص قيمة من نوع بيانات مختلف عن نوع المتغير ، فمثلا إذا كان x=1.0 يمكن أن تعطي خطأ وذلك إذا كان x معرف على أنه قيمة صحيحة int ، لذلك يجب أن يكوناً معرف على أنه double أي يقبل الكسور .

أيضا لاحظ أن اسم المتغير يجب أن يكون على اليسار بمعنى أن الجملة x = 1 تعتبر خطأ .

يمكن أن تحتوي جملة التخصيص على تعبير فمثلا :

area = radius * radius * 3.14159 ;

x = x + 3 ;

يمكن كتابة العملية الإسنادية السابقة بشكل مختصر x + = 3

واستعمال مثل هذه الاختصارات يسهل كثيرا جدا على المبرمجين كما يستطيع المترجم العمل أسرع عند ترجمة البرنامج .

والشكل التالي يوضح الاختصارات الشائعة الاستخدام في العمليات الإسنادية:

العامل	مثال	ما يقابله دون اختصار
+ =	c + = 7	c = c + 7
- =	d - = 4	d = d - 4
* =	e * = 5	e = e * 5
/ =	f = f / 3	f = f / 3
% =	g % = 9	g = g % 9

شكل (12) اختصارات العمليات الإسنادية

عامل الزيادة وعامل النقصان

تمدنا لغة الجافا بعامل الزيادة + + وعامل النقصان - - ويستطيع المبرمج زيادة قيمة متغير بمقدار الوحدة عن طريق استخدام عامل الزيادة + + مثل + + c وذلك بدلا من استخدام أي الجمل التالية

c + = 1; أو c = c + 1;

ويمكن كتابة عامل الزيادة أو عامل النقصان قبل أو بعد المتغير وذلك حسب الحاجة لاستخدامه داخل البرنامج.

والشكل التالي يوضح الفرق بين استخدام هذه العوامل قبل المتغير أو بعده

العامل	مثال	توضيح
+ +	+ + a	يتم زيادة المتغير a بمقدار 1 ثم تستخدم القيمة الجديدة للمتغير a في التعبير المتواجد فيه
+ +	+ + a	تستخدم القيمة الأولى للمتغير a في التعبير المتواجد فيه ثم بعد ذلك يتم زيادة المتغير a بمقدار 1
--	--b	يتم إنقاص المتغير b بمقدار 1 ثم تستخدم القيمة الجديدة للمتغير b في التعبير المتواجد فيه
--	b--	تستخدم القيمة الأولى للمتغير a في التعبير المتواجد فيه ثم يتم إنقاص المتغير b بمقدار ١

شكل (13) عامل الزيادة وعامل النقصان

الشكل التالي يوضح استخدام عامل الزيادة قبل أو بعد المتغير.

```
1. // Fig. 2.14  Increment.java
2. // Preincrementing and postincrementing
3.

4. public class Increment {
5. public static void main( String args[] )
6. {
7. int c;
8.
9. c = 5;
10. System.out.println( c );   // print 5
11. System.out.println( c++ ); // print 5 then postincrement
12. System.out.println( c );   // print 6
13.
14. System.out.println();      // skip a line
15.
16. c = 5;
17. System.out.println( c );   // print 5
18. System.out.println( ++c ); // preincrement then print 6
19. System.out.println( c );   // print 6
20. }
21. }
```

```
Output Window
5
5
6

5
6
6

Program Input:
```

شكل (2.14) مثال على استخدام عامل الزيادة وعامل النقصان

شرح البرنامج

السطر رقم 10

يتم طباعة قيمة المتغير c وهي 5

```
System.out.println( c );   // print 5
```

السطر رقم 11

```
system . out . println ( c + + ) ;
```

لاحظ أن عامل الزيادة أتى بعد المتغير لذلك يتم استخدام القيمة القديمة للمتغير c وهى 5 وتظهر 5 في جملة الطباعة ثم بعد الطباعة يتم زيادة المتغير c بمقدار ١ ويصبح c = 6

السطر 12

جملة طباعة للمتغير c بقيمة 6

```
System.out.println( c );   // print 6
```

السطر 14

```
System.out.println ( ) ;
```

طباعة سطر خالٍ

السطر 16

```
c = 5 ;
```

إعادة تخصيص القيمة 5 للمتغير c

السطر 17

طباعة قيمة المتغير c وهى 5

```
System.out.println( c );   // print 5
```

السطر 18

```
System.out.println( ++c ); // preincrement then print 6
```

لاحظ أن عامل الزيادة أتى قبل المتغير لذلك تتم أولا عملية الزيادة فيصبح المتغير c يساوي 6 ثم يتم استخدام القيمة الجديدة في جملة الطباعة فيطبع القيمة 6

السطر 19 جملة طباعة للمتغير c بالقيمة 6

```
System.out.println( c );   // print 6
```

العمليات الحسابية

معظم برامج الكمبيوتر تقوم بعمل عمليات حسابية والشكل التالي يوضح بعض العمليات الحسابية

العملية	العامل	التعبير الجبري	التعبير بالجافا
جمع	+	f+7	f+7
طرح	-	f-7	f-7
ضرب	❖	bm	b*m
قسمة	/	x	x/y
موديلاس	٪	r mod s	r%s

شكل (2.15) العمليات الحسابية

لاحظ الفرق بين التعبير الجبري والتعبير بواسطة الجافا وذلك في بعض العمليات مثل عملية الضرب والتي يعبر عنها بالعلامة (❖) ، عملية القسمة (/) ، أيضا عملية الموديلاس (٪) .

ملحوظة عمليات القسمة التي تتم على الأعداد الصحيحة تعطي عدداً صحيحاً فمثلا

خارج قسمة 4 / 7 هو 1

خارج قسمة 5 / 17 هو 3 وهكذا كما ترى يهمل الجزء العشري.

تمدنا الجافا بالمعامل (٪) الذي يعطي الباقي بعد القسمة ولاحظ أن هذه العملية تتم فقط على معاملات صحيحة فمثلا

ناتج العملية 4 % 7 هو 3

ناتج العملية 5 % 17 هو 2

تتعدد استخدامات المعامل (٪) كما سنرى إن شاء الله في الفصول القادمة.

خطأ شائع محاولة استخدام المعامل (٪) مع معاملات غير صحيحة يعطي syntax error

أولوية تنفيذ العمليات الحسابية

تستخدم الأقواس في الجافا تماما كما في الجبر العادي فمثلا

ضرب عدد a في حاصل جمع عددين تكتب هكذا a * (b+c)

تقوم الجافا بتنفيذ العمليات الحسابية بترتيب دقيق مشابه لما يحدث في الجبر:

١ - تقوم بحساب ما بداخل الأقواس أولا ، ولذلك تستخدم الأقواس من قبل المبرمج لحساب عملية معينة بترتيب معين يحدده المبرمج حسب ترتيب الأقواس.

٢ - إذا كانت هناك عملية بها عدة أقواس متداخلة يحسب أقصى قوس أولا من الداخل ثم الذي يليه إلي الخارج وهكذا ... .

٣ - يأتي بعد ذلك في الأولوية العمليات (الضرب ، القسمة ،الموديلاس) وهذه العمليات تعتبر في نفس درجة الترتيب. أما إذا كانت العملية الحسابية تحتوي على عدد من عمليات الضرب والقسمة و الموديلاس فإن التنفيذ يبدأ من اليسار إلى اليمين .

٤ - يأتي بعد ذلك الجمع والطرح وهما في نفس درجة الترتيب ولهذا مثل ما سبق إذا وجدت أكثر من عملية جمع وطرح في عملية حسابية واحدة فإن التنفيذ يبدأ من اليسار إلى اليمين .

مثال عبر عن العمليات الحسابية التالية بلغة الجافا وبين ترتيب تنفيذ العمليات داخل

y=mx+b
z= pr % q + w/x
y=x +bx +c

- التعبير بالجافا y=m*x +b;

لاحظ عدم وجود أقواس لذلك ينفذ الضرب أولا m*x ثم تنفذ بعد ذلك عملية الجمع .

- التعبير بالجافا z = p * r % q + w / x - y ;

```
z = p * r % q + w / x - y ;
    6   1   2   4   3   5
```

الأرقام السابقة تمثل ترتيب تنفيذ العمليات

لاحظ أن الضرب والقسمة و الموديلاس لها نفس مستوى الترتيب ولهذا يكون الترتيب من اليسار لليمين

-التعبير بالجافا y = a * x * x + b * x + c

```
y = a * x * x + b * x + c
  6   1   2   4   3   5
```

في المثال السابق بفرض القيم التالية :

a = 2 , b = 3 , c = 7 , x = 5

y = 2 * 5 * 5 + 3 * 5 + 7;

	2 * 5 = 10	الخطوة الأولى
y = 10 * 5 + 3 * 5 + 7;		
	10 * 5 = 50	الخطوة الثانية
y = 50 + 3 * 5 + 7;		
	3 * 5 = 15	الخطوة الثالثة
y = 50 + 15 + 7;		
	50 + 15 = 65	الخطوة الرابعة
y = 65 + 7;		
	65 + 7 = 72	الخطوة الخامسة
y = 72;		الخطوة السادسة

كما في التعبير الجبري من المفضل استعمال الأقواس حتى غير الضرورية وذلك لجعل التعبير أوضح

Y = (a * x * x) + (b * x) + c ;

العاملات المنطقية

يدرج الجدول التالي المعاملات المنطقية شائعة الاستخدام في برمجة java

العامل	العملية
&&	And المنطقية
||	Or المنطقية
!	Not المنطقية

شكل (2.16) العاملات المنطقية

يتم استخدام العاملات المنطقية مع المعاملات التي يكون لها إحدى قيم true أو false ،أو التي تكون قيم يمكن تحويلها إلى true أو false .

يقوم العامل المنطقي && بتقييم اثنين من المعاملات وإرجاع القيمة true إذا كان كلا المعاملين true . فيما عدا ذلك ، سيرجع العامل && القيمة false .

يتم استخدام ذلك في التفرع الشرطي حيث يتم تحديد اتجاه برنامج java عن طريق اختبار اثنين من الشروط .إذا تم تلبية كلا الشرطين ، سيتوجه البرنامج إلى اتجاه معين ، فيما عدا ذلك ، سيأخذ البرنامج اتجاه مختلف .

على العكس من العامل && الذي يحتاج كلا المعاملين لكي يكون true ، يقوم العامل || بتقييم معامليه ويرجع القيمة true . اذا لم يرجع أي من المعاملين القيمة true ،فإن العامل سيرجع القيمة false يمكن الاستفادة من ذلك في برمجة java لأداء إجراء معين إذا لم تتم تلبية أيا من شرطي الاختبار .

يعتبر العامل المنطقي الثالث ! عامل أحادي يتم استخدامه قبل معامل واحد . يقوم هذا العامل بإرجاع القيمة العكسية للمعامل المعطى ، وبذلك ، إذا كان المتغير a له القيمة true ، فإن a! سيكون له القيمة false . في برنامج java ، من الأفضل تبديل قيمة المتغير في التكرارات المتتالية للحلقة باستخدام عبارة مثل a=!a . يؤكد ذلك أنه في كل دورة سيتم تغيير القيمة .

أمثلة على العاملات المنطقية

يوضح البرنامج التالي الكيفية التي يمكن بها اختبار القيم المنطقية باستخدام المعاملات المنطقية التي سبق عرضها :

```
1.  // Fig.  2.17    : Logical.java
2.  // Logical Operator
3.  public class Logical
4.  {
5.  public static void main ( String [] args )
6.  {
7.  //declare & initialize test variable
8.  boolean a  = true , b = false ;
9.  boolean c1 =(a && a);      // test if both are true
10. boolean c2 =(a && b);
11. boolean c3 =(b && b);
12.
13. boolean c4 =(a || a) ;       //test if either is true
14. boolean c5 =(a || b);
15. boolean c6 =(b || b);
16.
17. boolean c7 = !a;            // invert initial values
18. boolean c8 = !b:
19.
20. // display the results
21. System.out.println (“and:\n1:”+c1+”2:”+c2+”3:”+c3);
22. System.out.println (“or:\n4:”+c4+”5:”+c5+”6:”+c6);
23. System.out.println (“not:\n7:”+c7+”8:”+c8);
24. }
25. }
```

```
Output Window
and:
1:true2:false3:false
or:
4:true5:true6:false
not:
7:false8:true

Program Input:
```

شكل(2.17) مثال على العمليات المنطقية

شرح البرنامج

السطر رقم 8 تم تعريف متغيرين من النوع boolean هما a ، b وإعطائهما قيم ابتدائية هي true، false على الترتيب.

السطور 9,10,11

تم تعريف المتغيرات c1,c2,c3 وذلك لمناقشة الحالات المختلفة للعملية &&

السطور 13,14,15

تم تعريف المتغيرات c4,c5,c6 وذلك لمناقشة الحالات المختلفة للعملية ||

السطور 17,18

تم تعريف المتغيرين c7,c8 وذلك لمناقشة الحالات المختلفة للعملية !

السطور 21,22,23

هي جمل لطباعة المتغيرات c1,c2,c3,c4,c5,c6,c7,c8.

اتخاذ القرار: التساوي والعمليات العلاقية

سوف نناقش في هذا الجزء جملة if في أبسط صورها والتي تسمح للبرنامج بأخذ قرار معين معتمد على صحة أو خطأ شرط ما .

إذا تحقق الشرط نفذت الجملة التالية لجملة if أما إذا لم يتحقق لم تنفذ الجملة التالية لجملة if .

ملحوظة: في الحقيقة جملة **if** والجملة التي تليها هما جملة واحدة.

الشروط في جملة if تكتب باستخدام عمليات التساوي والعمليات العلاقية والجدول التالي يوضح جميع عمليات التساوي والعمليات العلاقية في لغة الجافا:

شكل عمليات التساوي في الجبر	شكل عمليات التساوي في الجافا	مثال على الشرط في الجافا	معنى الشرط
=	==	x==y	x تساوي y
≠	!=	x!=y	x لا تساوي y

شكل (2.18) عمليات التساوي

شكل العمليات العلاقية في الجبر	شكل العمليات العلاقية في الجافا	مثال على الشرط في الجافا	معنى الشرط
>	>	x>y	x أكبر من y
<	<	x<y	x أقل من y
≥	>=	x>=y	x أكبر من أو تساوي y
≤	<=	x<=y	x أقل من أو تساوي y

شكل (2.19) العمليات العلاقية

خطأ شائع:

عند كتابة العمليات != ، == ، >= ، <= ، وبينهما مسافة مثل = = ، > = ، < = ، ! = يعطي البرنامج syntax error.

أيضا عند عكس رموز العملية الواحدة مثل =< ، => ، =! يعطي أيضا syntax error

المثال التالي نستخدم 6 جمل شرطية if للمقارنة بين رقمين مدخلين بواسطة المستخدم وذلك كتطبيق على عمليات التساوي والعمليات العلاقية.

في هذا المثال يقوم المستخدم بإدخال قيمتين من خلال صندوق حوار ، بعد ذلك يقوم البرنامج بتحويل هاتين القيمتين إلى عددين صحيحين ثم يقوم بتخزين الرقمين في المتغيرين number2 ،number1. ثم يقوم البرنامج بعمل المقارنات عن طريق جمل if ويعرض الناتج في صندوق المعلومة .

الشكل التالي يوضح البرنامج ونموذج لشكل المخرجات.

```
1.  // Fig. 2.20: Comparison.java
2.  // Compare integers using if structures, relational operators
3.  // and equality operators.
4.
5.  // Java extension packages
6.  import javax.swing.JOptionPane;
7.
8.  public class Comparison {
9.
10. // main method begins execution of Java application
11. public static void main( String args[] )
12. {
13. String firstNumber;
14. String secondNumber;
15. String result;
16. int number1;
17. int number2;
18.
19. // read first number from user as a String
20. firstNumber =
21. JOptionPane.showInputDialog( "Enter first integer:" );
22.
23. // read second number from user as a String
24. secondNumber =
25. JOptionPane.showInputDialog( "Enter second integer:" );
26.
27. // convert numbers from type String to type int
28. number1 = Integer.parseInt( firstNumber );
29. number2 = Integer.parseInt( secondNumber );
30.
31. // initialize result to empty String
32.
33. result = "";
34. if ( number1 == number2 )
35. result = number1 + " == " + number2;
36.
37. if ( number1 != number2 )
38. result = number1 + " != " + number2;
39.
```

```
40. if ( number1 < number2 )
41. result = result + "\n" + number1 + " < " + number2;
42.
43. if ( number1 > number2 )
44. result = result + "\n" + number1 + " > " + number2;
45.
46. if ( number1 <= number2 )
47. result = result + "\n" + number1 + " <= " + number2;
48.
49. if ( number1 >= number2 )
50. result = result + "\n" + number1 + " >= " + number2;
51.
52. // Display results
53.
54. JOptionPane.showMessageDialog(
55. null, result, "Comparison Results",
56. JOptionPane.INFORMATION_MESSAGE );
57. System.exit( 0 );  // terminate application
58.
59. }  // end method main
60.
61. }  // end class Comparison
```

Input
Enter first integer:
111
OK Cancel

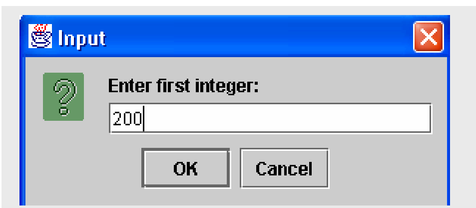
Input
Enter first integer:
111
OK Cancel

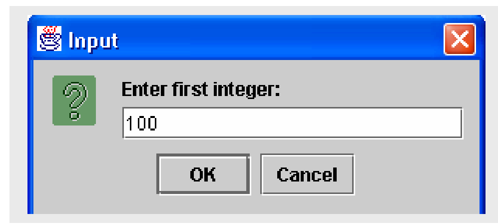
Comparison Results
111 == 111
111 <= 111
111 >= 111
OK

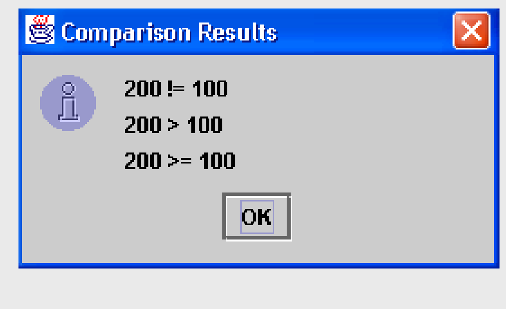
Input
Enter first integer:
100
OK Cancel

Input
Enter first integer:
200
OK Cancel

Comparison Results
100 != 200
100 < 200
100 <= 200
OK







شكل (2.20) مثال على عمليات المقارنة

شرح البرنامج

من خلال البرنامج يتضح أن فصل التطبيق Comparison بدأ تعريفه في السطر 8

```
public class Comparison {
```

كما قلنا سابقا فإن الطريقة main و التي كتبت من السطر 11 إلى السطر 59 يبدأ تنفيذها أولا في كل تطبيقات الجافا.

السطور من 13 إلى 17

```
String firstNumber;
String secondNumber;
String result;
int number1;
int number2;
```

هي عبارة عن تعريف للمتغيرات المستخدمة في الطريقة main

لاحظ أن هناك ٣ متغيرات من النوع String

firstNumber وهو الرقم الأول المدخل من المستخدم في صندوق الحوار ويدخل على شكل نص

secondNumber هو الرقم الثاني المدخل من المستخدم في صندوق الحوار ويدخل على شكل نص

result وهو متغير يخزن فيه الناتج على صورة نص

هناك أيضا ٢ متغير من النوع int

number1 وهو الرقم الأول الذي سيتم مقارنته

number2 وهو الرقم الثاني الذي سيتم مقارنته

لاحظ أن المتغيرات ذات النوع الواحد يمكن تعريفها في نفس السطر

مثال :

```
String firstNumber , secondNumber , result;
```

السطور 20-21

```
firstNumber =
JOptionPane.showInputDialog( "Enter first integer:" );
```

هذا الأمر يسمح للمستخدم بإدخال رقم داخل صندوق الحوار ويخزن داخل المتغير firstNumber على شكل نص string

السطور 24-25

```
secondNumber =
JOptionPane.showInputDialog( "Enter second integer:" );
```

هذا الأمر يسمح للمستخدم بإدخال الرقم الثاني من خلال صندوق حوار ويخزن في المتغير secondNumber على شكل نص String

السطور 28-29

```
number1 = Integer.parseInt( firstNumber );
number2 = Integer.parseInt( secondNumber );
```

كل من السطرين السابقين يقوم بتحويل قيمة نص String إلى قيمة صحيحة int ففي السطر 28 قمنا بتحويل قيمة المتغير firstNumber النصية إلى قيمة صحيحة ويخزنها في المتغير number1

أيضا السطر 29 حولت قيمة المتغير secondNumber النصية إلى قيمة صحيحة وخزنت في المتغير number2 .

السطر رقم 33

```
result = "";
```

في هذا السطر قمنا بتخصيص نص فارغ empty string إلى المتغير result وذلك لأن كل متغير يعرف داخل طريقة method لابد أن يأخذ قيمة ابتدائية قبل أن يستخدم لذلك تم تخصيص هذا النص الفارغ مؤقتا كقيمة ابتدائية. وهنا يجب التنويه عن هذا الخطأ الشائع الحدوث :

خطأ شائع : عدم إعطاء المتغير المعرف داخل طريقة method قيمة ابتدائية وذلك قبل استخدامه **داخل الطريقة يعطى syntax error**

السطر 34-35

```
if ( number1 == number2 )
result = number1 + " == " + number2;
```

هذه هي جملة if وعادة تبدأ جملة if بكلمة if يتبعها شرط داخل أقواس ثم يأتي بعد ذلك جملة هي في الواقع من التركيب الأساسي ل if لذلك كان يمكن كتابة السطر ٣٥ مع السطر ٣٤ دون فصلهما ولكن تم الفصل للسهولة أثناء القراءة ، ولذلك نلاحظ عدم وجود الفاصلة المنقوطة (;) والتي تدل على نهاية الجملة في السطر 34 ، معنى ذلك أن الجملة لم تنتهِ عند الشرط .

نفرض أن number1، number2 متساويين إذن تنفذ الجملة التالية

```
result = result + number1 + "==" + number2;
```

وفي هذا السطر تم تخصيص

```
result + number1 + "==" + number
```

للمتغير result.

وهنا نلاحظ وجود قيمتين صحيحتين هما number1، number2 فكيف يتم إضافة قيمة صحيحة إلى أخرى نصية string ثم تخزين الناتج في متغير أيضا نص string

في حقيقة الأمر هذه العملية تسمى string concatenation

يتم تحويل قيم number1، number2 إلى قيم نصية ثم تضاف إلى القيمة وتخزن الناتج في المتغير result.

أخطاء شائعة :

- عند تبديل العملية (=) مكان (==) في شرط جملة if يعطى syntax error
- وضع فاصلة منقوطة بعد الشرط مثل ; if (number1 == number2)

يعطى خطأ منطقي أي خطأ يظهر أثناء تنفيذ البرنامج وذلك لأنه يعتبر أن جواب الشرط جملة خالية.

أسئلة وتمارين

١) حدد أياً من الجمل التالية صح وأيها خطأ مع التعليل

ا - عند تنفيذ برنامج ما ، إذا وجدت ملاحظات فإن الكمبيوتر يقوم بطباعة النص الذي يأتي بعد // على الشاشة. ( )

- -

- -

ب - العامل موديلاس ٪ يمكن استخدامها فقط مع معاملات صحيحة ( )

- -

- -

ج - العمليات الحسابية التالية تأتي كلها في نفس مستوى أولوية التنفيذ - , + ,%, / , * ( )

- -

- -

د - الطريقة Integer.parseInt تحول رقماً صحيحاً int إلى نص string ( )

- -

- -

٢) أكتب جمل الجافا التي تحقق كل من المهام التالية:

أ - حول قيمة نصية String إلى قيمة صحيحة integer ثم خزن القيمة المحولة في متغير صحيح age ، وذلك بفرض أن النص مخزن في value .

- -
- -
- -

ب - إذا كان المتغير number لا يساوي 10 اعرض في صندوق الرسالة التالية:

"The variable number is not equal to 10"

- -
- -
- -

٣) حدد ما هو الخطأ في الجمل التالية:

ا -

```
if ( c < 7 );
 JoptionPane.showMessageDialog (null, "c is less than 7");
```

- -
- -
- -

ب –

```
  if ( c => 7 )
  JoptionPane.showMessageDialog (null,"c is equal or greater than
7");
```

- -
- -

٤ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بسؤال المستخدم أن يُدخل رقمين، ثم يقوم بعد ذلك بطباعة الرقم الأكبر متبوعاً بالنص التالي " is larger " وذلك داخل صندوق رسالة. وإذا كانا الرقمان متساويين يطبع الرسالة " these number are equal ".

٥ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة ثلاثة أرقام صحيحة من المستخدم، ثم يقوم بعرض المجموع، المتوسط الحسابي، حاصل الضرب، الرقم الأصغر والرقم الأكبر وذلك داخل صندوق رسالة.

٦ -اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة نصف قطر دائرة من المستخدم، ثم يقوم بعد ذلك بطباعة قطر الدائرة، المحيط والمساحة.

افرض أن الثابت الطبيعي $\pi = 3.14159$

- ملحوظة: يمكنك استخدام الثابت Math. PI وذلك للثابت الطبيعي وهذه القيمة تعتبر أدق من القيمة 3.14159

٧ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة 5 أرقام صحيحة من المستخدم، ثم يقوم بطباعة الرقم الأكبر والرقم الأصغر.

- ملحوظة: استخدم التقنيات التي تعلمتها في هذا الفصل فقط.

٨ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة عدد صحيح من المستخدم، ثم يقوم بطباعة رسالة يحدد فيها ما إذا كان هذا العدد فردياً أم زوجياً.

- ملحوظة: استخدم عامل الموديلاس، أي عدد زوجي هو مضاعفات الرقم 2 لذلك أي مضاعف للعدد 2 يعطي باقياً 0 عند القسمة على 2.

٩ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة عددين صحيحين من المستخدم، ثم يحدد ويطبع ما إذا كان الأول هو مضاعف الثاني.

١٠ - اكتب برنامجاً تطبيقياً بلغة الجافا يقوم بقراءة عدد من المستخدم مكون من 5 أرقام صحيحة، ثم يقوم البرنامج بعد ذلك بطباعة الأرقام وبين كل رقم وآخر مسافة.

مثال: إذا كان العدد هو 42339 يقوم البرنامج بطباعة 9 3 3 2 4

المملكــة العربيــة الســعودية
المؤسسة العامة للتعليم الفني والتدريب المهني
الإدارة العامة لتصميم وتطوير المناهج



برمجه الحاسب

أدوات التحكم البنائي

أدوات التحكم البنائي



الجدارة:

أن يكون المتدرب قادرا على كتابة الشفرة البرمجية code للبرامج المتوسطة نسبيا

الأهداف :

عندما تكمل هذه الوحدة يكون لديك القدرة على:

١. فهم ومعرفة جمل التفريع
٢. كتابة جمل if ، if/else ، if المتداخلة
٣. كتابة جملة switch
٤. معرفة الحلقات التكرارية
٥. كتابة واستخدام حلقة while
٦. كتابة واستخدام حلقة do/while
٧. كتابة واستخدام حلقة for
٨. كتابة واستخدام الحلقات المتداخلة
٩. كتابة برامج متوسطة

مستوي الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة 100 ٪

الوقت المتوقع للتدريب : ١٦ ساعة

الوسائل المساعدة :

- حاسب إلى
- قلم
- دفتر

متطلبات الجدارة :

اجتياز جميع الحقائب السابقة

أدوات التحكم البنائي

عادة يتم تنفيذ الجمل في البرامج بطريقة تتابعيه جملة بعد أخرى ولكن عند استخدام أداة من أدوات التحكم مثل if لا تنفذ الجملة إلا إذا تحقق الشرط السابق.

- في الفصل السابق درسنا شكلاً من أشكال اتخاذ القرار عن طريق استخدام جملة if في أبسط صورها وقلنا

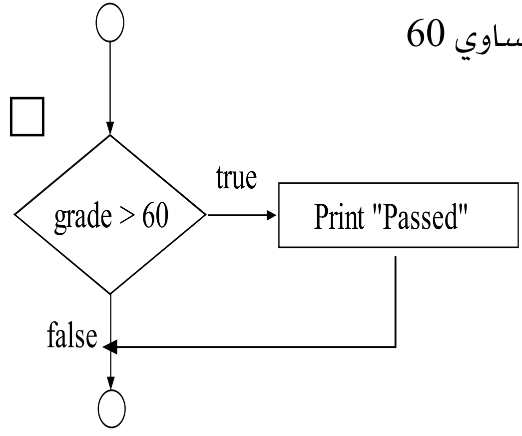
```
If (number1 == number2 )
result = result+ " number1 == number2 " ;
```

جملة if السابقة تسمى if selection structure حيث إنها تقوم بانتخاب الجملة التي يتم تنفيذها

فمثلا : افرض أن درجة اجتياز أي اختبار هي 60 من 100 تكتب جملة if على النحو

```
If (studentGrade > = 60 )
System.out.println (" passed" );
```

وبذلك نرى أن كلمة (" passed ") لن يتم طباعتها على الشاشة إلا إذا كانت درجة الطالب في الاختبار أكبر من أو تساوي 60

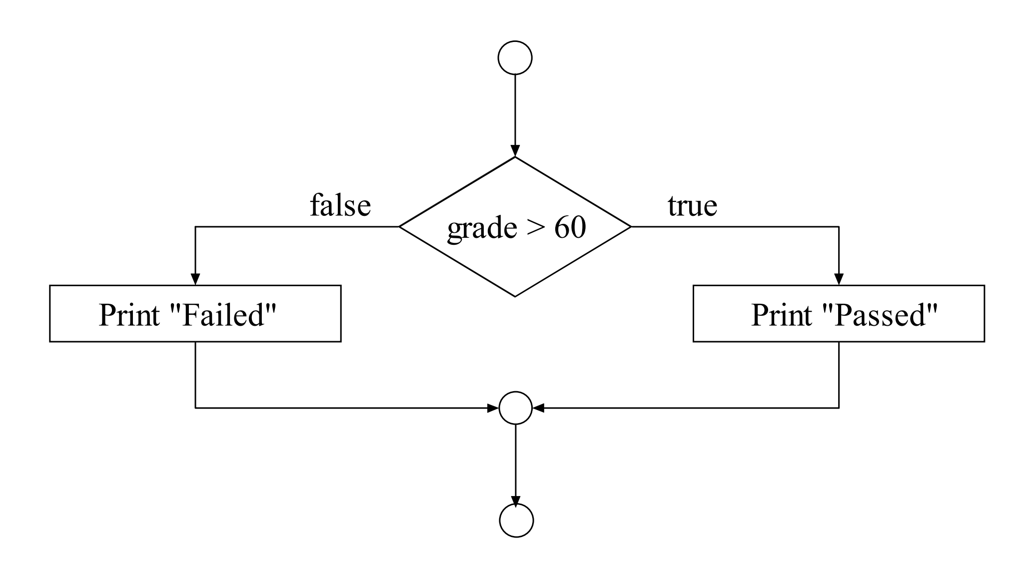


شكل (1-3) خريطة تدفق لجملة if

من الشكل السابق يتضح أن

- إذا تحقق الشرط نفذت الجملة طباعة كلمة (" passed ") ثم تنفيذ باقي جمل البرنامج
- إذا لم يتحقق الشرط لم تنفذ الجملة وأكمل تنفيذ باقي جمل البرنامج.

جملة if \ else



شكل (3-2) خريطة تدفق لجملة if/else

يمكننا استخدام جملة if \ else إذا أردنا

- عند تحقق الشرط تنفذ جملة ما ثم ينفذ باقي البرنامج
- عند عدم تحقق الشرط تنفذ جملة أخرى ثم بعدها ينفذ باقي البرنامج

مثال

```
if ( studentGrade > = 60 )
System.out.println ( “ passed “ );
else
System.out.println ( “ failed “ );
```

من هذا المثال يتضح أنه إذا كان تقدير الطالب أكبر من أو يساوي 60 طبعت على الشاشة كلمة " passed" أما إذا كانت الدرجة غير ذلك أي أقل من 60 طبعت كلمة " failed "

العملية (? :)

يمكن في لغة الجافا اختصار جملة if \ else بالعملية (?:) وبذلك يمكن التعبير عن جملة if \ else السابقة كالتالي

```
System.out.println (studentGrade >= 60 ? “ passed “ : “ failed “ ) ;
```

لاحظ أن **?** تمثل جملة if ، **:** تمثل else

جمل if \ else المتعددة

يمكن استخدام جمل if \ else المتعددة وذلك في حالات الاختيار من متعدد

مثال : المطلوب طباعة a إذا كانت درجة الاختبار أكبر من أو تساوي 90 ، b إذا كانت الدرجة أكبر من أو تساوي 80 ، c إذا كانت الدرجة من 70 إلى 79 ، d إذا كانت الدرجة من 69 إلى 60 ويطبع f غير ذلك :

```
if (studentGrade >= 90 )
System.out.println ( " a " );
else if ( studentGrade >= 80 )
System.out.println ( " b " );
else if ( studentGrade >= 70 )
System.out.println ( " c " );
else if ( studentGrade >= 60 )
System.out.println ( " d " );
else
System.out.println ( " f " );
```

لاحظ أنه عند تحقق الشرط في جملة من الجمل السابقة تنفذ الجملة التالية فقط .

أما إذا أردنا تنفيذ أكثر من جملة لابد من عمل أقواس { }

فمثلا إذا أردنا في المثال السابق أن يطبع f وجملة " you must take this course again "

فتكون الجملة الأخيرة على الشكل

```
else
    {
        System.out.println ( " f " );
        System.out.println ( " you must take this course again " );
    }
```

لاحظ أنه عند وجود الأقواس يتم التعامل مع ما بداخلها على أنها جملة واحدة

خطأ شائع :

عند نسيان أحد الأقواس أو كليهما يؤدي إلى وجود خطأ في بناء الجملة syntax error أو خطأ منطقي ففي المثال السابق عند نسيان أحد الأقواس يعطي syntax error أما عند نسيان القوسين معا فإنه لا يعطي خطأ عند عمل ترجمة compile ولكن يعطي خطأ منطقياً أي عند التنفيذ إذا كانت الدرجة أقل من 60 يطبع f ولكن لا يطبع الجملة " you must take this course "

عند وضع فاصلة منقوطة (;) بعد الشرط يعطي خطأ منطقي logical error إذا كنا نستخدم جملة if البسيطة ، أما إذا كنا نستخدم جمل if المتعددة يعطي syntax error

استخدام جملة switch

في حالات الاختيار من متعدد يمكن أن تحل جملة switch محل if/else المتعددة

تكتب جملة switch على الشكل التالي:

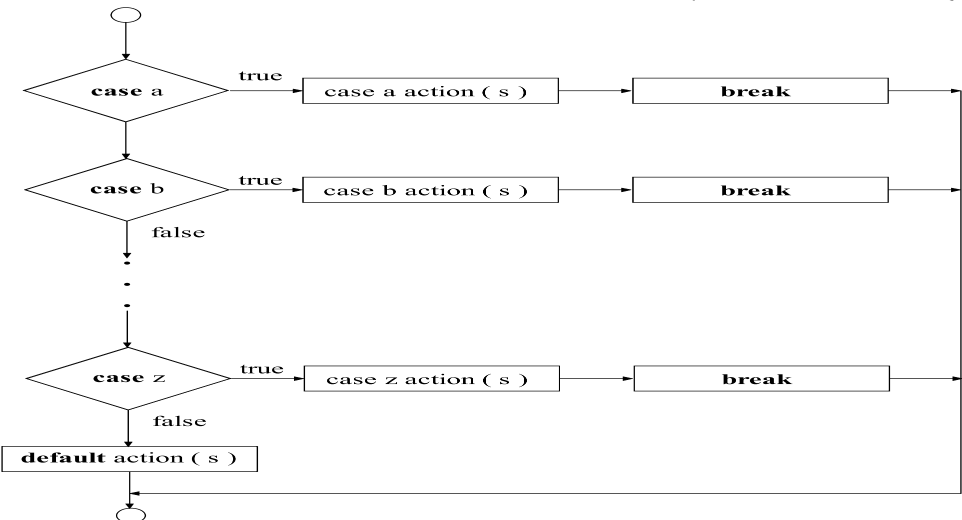
```
switch (switch-expression)
{ case value1: statement(s)1;break;
  case value2: statement(s)2;break;
  …
  case valueN: statement(s)N;break;
        default :           statement(s)- for – default;
}
```

لاحظ أن switch expression لابد أن تكون قيمة (charأو byte أو short أوint)

لاحظ أيضا أن قيم value1,value2,…,valueN لابد أن تكون من نفس نوع switch expression

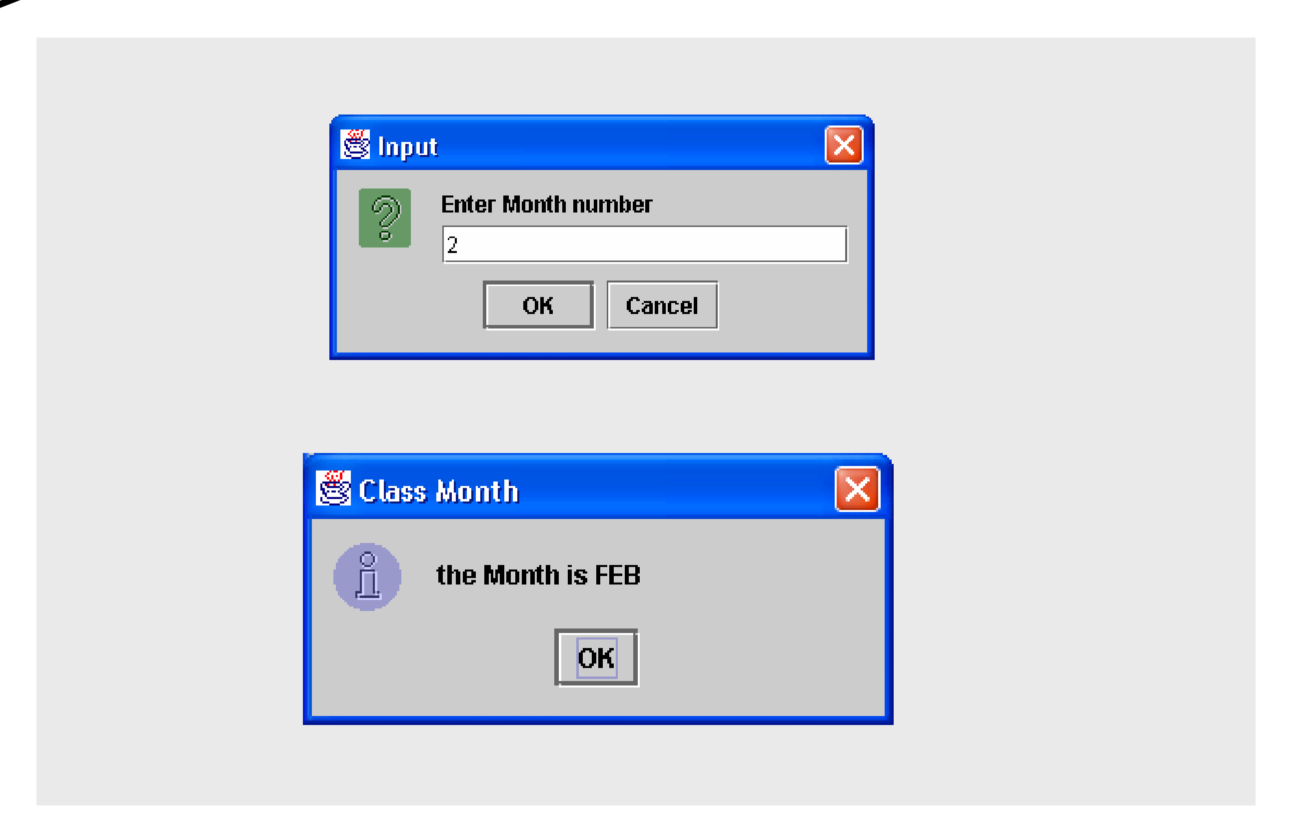
الجملة(الجمل) statement(s) تنفذ إذا وإذا فقط كانت القيمة value تساوي switch expression ، ثم تأتي بعد ذلك جملة break لكي تعمل هروب من جملة switch

الحالة الافتراضية default وهي اختيارية ويمكن أن تستخدم لتنفيذ مهمة في حالة عدم تحقق أي حالة ودائما تكون هذه الجملة في آخر جملة switch .



شكل (3-3) خريطة تدفق جملة switch

```
1.  // Fig. 3.4 : Month.java
2.  // Class Month program with switch statements.
3.  // Java extension packages
4.  import javax.swing.JOptionPane;
5.
6.  public class Month {
7.  // main method begins execution of Java application
8.  public static void main( String args[] )
9.  {
10. int month;  // number of month number entered
11. String input;      // month number typed by user
12. String name;        // name of month
13.
14. // Processing phase
15. // prompt for input and read Month number from user
16. input = JOptionPane.showInputDialog(
17. "Enter Month number" );
18. // convert grade from a String to an integer
19. month = Integer.parseInt( input );
20. switch ( month )
21. {
22. case 1:name="JAN";break;
23. case 2:name="FEB";break;
24. case 3:name="MAR";break;
25. case 4:name="APR";break;
26. case 5:name="MAY";break;
27. case 6:name="JUN";break;
28. case 7:name="JUL";break;
29. case 8:name="AUG";break;
30. case 9:name="SEP";break;
31. case 10:name="OCT";break;
32. case 11:name="NOV";break;
33. case 12:name="DEC";break;
34.
35. default :name=" invalid Month number ";
36. }
37.
38. // display name of month number
39. JOptionPane.showMessageDialog( null,
40. "the Month is " + name ,
41. "Class Month", JOptionPane.INFORMATION_MESSAGE );
42.
43. System.exit( 0 );    // terminate application
44.
45. }  // end method main
46.
```



شكل (3-4) مثال على جملة switch

شرح البرنامج

البرنامج يقوم بقراءة رقم من المستخدم ثم يقوم بطباعة اسم الشهر المقابل له لاحظ وجود جملة switch في السطور من ٢٠ إلى ٣٦ في حالة تطابق الرقم المدخل مع أي حالة من حالات جملة switch يتم تنفيذ الجملة التي تلي الحالة فمثلا إذا كان الرقم المدخل هو ٢ فهذا الرقم يتطابق مع الحالة :case 2 لذلك يتم تنفيذ الجملة التالية وهي;name="FEB" وهي تخصيص السلسلة FEB إلى المتغير name .

لاحظ وجود جملة ;break والتي تتسبب في الهروب من جملة switch .

في حالة عدم تطابق الرقم المدخل من المستخدم مع الحالات الموجودة يتم تنفيذ الحالة الافتراضية الموجودة في السطر رقم 35

```
default :name=" invalid Month number ";
```

تخصيص النص السابق للمتغير name .

السطر ٣٦ يحتوي على } وهي تمثل نهاية جملة switch.

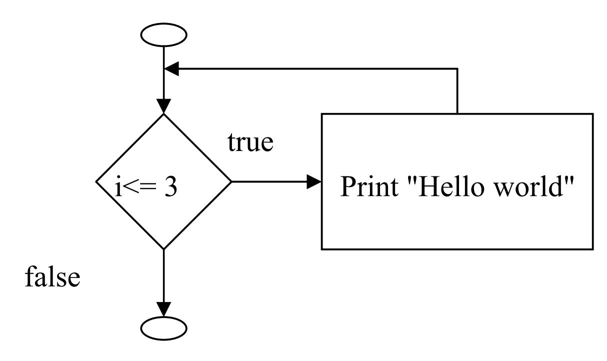
السطور من ٣٩ إلى ٤١

```
JOptionPane.showMessageDialog( null,"the Month is " + name ,
"Class Month", JOptionPane.INFORMATION_MESSAGE );
```

جملة طباعة للمتغير في صندوق رسالة

بناء حلقة while التكرارية

تسمح الجمل التكرارية للمبرمج أن يعرف جملة ما أو عدة جمل أن يحدث لها تكرار طالما أن الشرط صحيح



شكل (3-5) خريطة تدفق حلقة while

مثال إذا أردنا طباعة جملة " hello world " ٣ مرات

```
int   i= 1 ;
While   ( i < = 3 )
{
System.out.println  ( " hello world " ) ;
i + = 1 ;
}
```

نجد من المثال السابق أن جملة التكرار while لها ٣ بنود

١ - جملة الإشعال : وهي تعريف المتغير و إعطاؤه قيمة ابتدائية هي ١
٢ - الشرط : وهو شرط حدوث التكرار وهو أن تكون i أقل من أو تساوي ٣
٣ - زيادة العداد : وفي المثال السابق تتم زيادة i بمقدار ١ بعد جملة الطباعة

لاحظ وجود الأقواس وذلك لأننا نريد تكرار أكثر من جملة وهي جملة الطباعة وجملة زيادة العداد.

مثال

```
int sum = 0 ;
int i = 1
While ( i < = 10 )
{       sum + = i ;
        i + = 1 ;
}
System.out.println ( sum )
```

وفي المثال السابق أردنا طباعة مجموعة الأعداد من ١ إلى ١٠

لاحظ وجود جملة الطباعة خارج الأقواس .

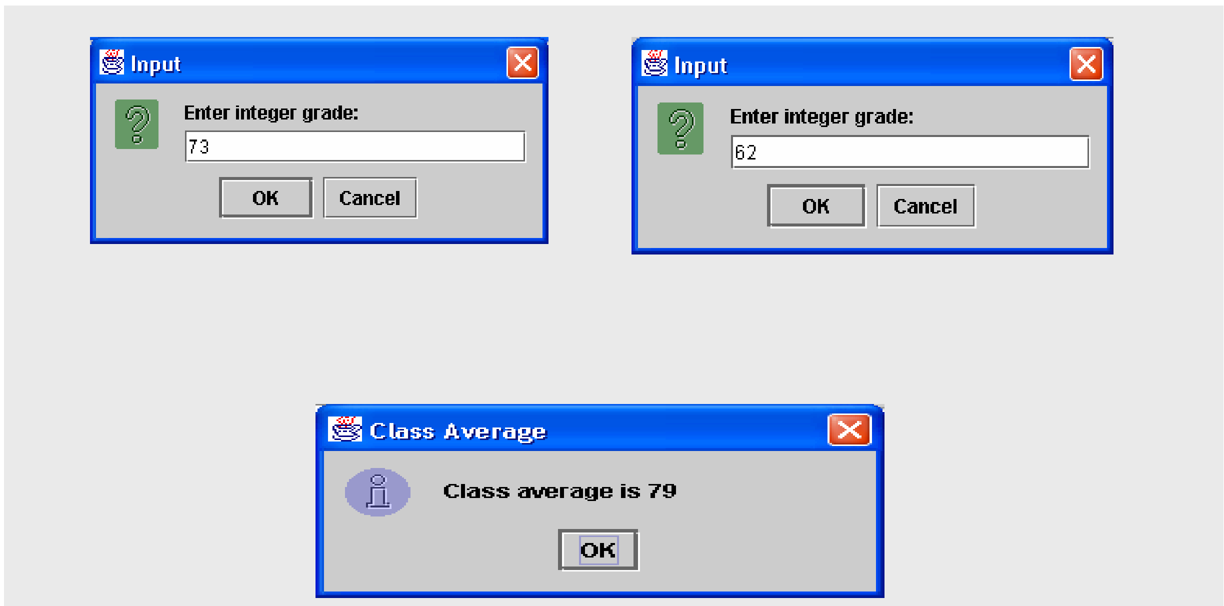
خطأ شائع : عدم وضع جملة تجعل الشرط لجملة while خطأ يعتبر خطأ منطقياً لأنه يجعل الحلقة تستمر إلى ما لانهاية وذلك لأن الشرط يبقى دائما صحيحاً.

فمثلا في المثال السابق عند إلغاء جملة زيادة العداد تظل دائما قيمة المتغير i هي ١ ويستمر الشرط صحيحاً وتستمر الحلقة إلى ما لانهاية.

- أيضا كتابة while بحرف كبير مثل While يعتبر خطأ في بناء الجملة ، وتذكر أن لغة الجافا تفرق بين الحروف الكبيرة والصغيرة وأن جميع الكلمات المحجوزة reserved keywords مثل if – else – while - ……… تحتوي كلها على حروف صغيرة .

مثال

```
1. // Fig. 3.6: Average1.java
2. // Class average program with counter-controlled repetition.
3. // Java extension packages
4. import javax.swing.JOptionPane;
5.
6. public class Average1 {
7. // main method begins execution of Java application
8. public static void main( String args[] )
9. {
10. int total,          // sum of grades input by user
11. gradeCounter,   // number of grades entered
12. gradeValue,     // grade value
13. average;        // average of all grades
14. String grade;       // grade typed by user
15.
16. // Initialization Phase
17. total = 0;          // clear total
18. gradeCounter = 1;   // prepare to loop
19.
20. // Processing Phase
21. while ( gradeCounter <= 10 ) {  // loop 10 times
22.
23. // prompt for input and read grade from user
24. grade = JOptionPane.showInputDialog(
25. "Enter integer grade: " );
26. // convert grade from a String to an integer
27. gradeValue = Integer.parseInt( grade );
28.
29. // add gradeValue to total
30. total = total + gradeValue;
31.
32. // add 1 to gradeCounter
33. gradeCounter = gradeCounter + 1;
34.
35. }  // end while structure
36.
37. // Termination Phase
38. average = total / 10;  // perform integer division
39.
40. // display average of exam grades
41. JOptionPane.showMessageDialog( null,
42. "Class average is " + average, "Class Average",
43. JOptionPane.INFORMATION_MESSAGE );
44. System.exit( 0 );     // terminate the program
45. }  // end method main
46. }  // end class Average1
```



شكل (3-6) مثال عل استخدام حلقة while

شرح البرنامج

المثال السابق يقوم بأخذ تقديرات الطلاب في مادة ما ويقوم بطباعة المتوسط الحسابي للتقديرات.

السطر رقم ٤ ; import javax.swing.JOptionPane

تقوم هذه الجملة بعمل استيراد للفصل JOptionPane وذلك لكي يسمح للبرنامج بقراءة البيانات من لوحة المفاتيح ، أيضا يسمح بعرض ناتج التنفيذ على الشاشة وذلك من خلال صندوق حوار السطر رقم 6 public class Averagel {

تعريف اسم الفصل Averagel تذكر أن أي تطبيق لابد أن يحتوى على الطريقة main لكي يتم تنفيذ التطبيق الطريقة main في هذا المثال من (السطر 8 إلى 45) السطور من 10 إلى 14 تقوم بتعريف المتغيرات

total , gradeCounter , gradeValue , average

من النوع الصحيح integer

أيضا تقوم بتعريف المتغير grade من نوع متسلسلة String والذي يقوم بحفظ النص الذي يدخله المستخدم في صندوق الحوار

المتغير gradeValue يحفظ القيمة الصحيحة للنص المخزن في المتغير grade وذلك بعد عملية التحويل التي سيجريها البرنامج على النص.

لاحظ أن كل المتغيرات السابقة تم تعريفها داخل الطريقة main وهي متغيرات محلية أي تستخدم فقط داخل هذه الطريقةmain ولا يمكن التعامل معها من طريقة أخرى .

أيضا لا يجب استخدام متغير ما قبل تعريفه.

السطور من 17 , 18

```
total = 0;          // clear total
gradeCounter = 1;   // prepare to loop
```

هي جمل تخصيص تقوم بإشعال المتغيرين gradeCounter , total وإعطائهما قيمتين ابتدائيتين 0، 1 على الترتيب . وذلك قبل استخدامها في حسابات البرنامج

السطر 21

```
while ( gradeCounter < = 10 ) {
```

بداية حلقة while التكرارية وتحدد شرط استمرار الحلقة وهو أن تكون gradeCounter أقل من أو تساوي 10 .

لاحظ وجود القوس } وهو يمثل بداية الجمل المراد تكرارها داخل الحلقة .

خطأ شائع : استخدام المتغير في الحسابات قبل إعطائه قيمة ابتدائية يظهر رسالة الخطأ التالية

variable may not have been initialized

وهو خطأ يوضح أن المتغير يجب إشعاله وإعطاؤه قيمة ابتدائية قبل استخدامه ، ففي المثال السابق إذا أهملنا السطر رقم ١٨ الخاص بإشعال المتغير gradeCounter ، ثم نستخدم نفس المتغير في السطر ٢١ في الشرط لا يمكن للبرنامج من تحديد قيمة gradeCounter وبالتالي يعطي خطأ السطور 24، 25

```
grade = JOptionPane.showInputDialog("Enter integer grade: " );
```

تقوم بعرض صندوق حوار للمستخدم يطلب من إدخال تقدير الطالب ،تخزن القيمة المدخلة في المتغير grade يتم تحويل النص إلى عدد صحيح ويحفظ في المتغير gradeValue عن طريق السطر 27

```
gradeValue = Integer . parseInt ( grade ) ;
```

تذكر أن الفصل Integer يوجد داخل الحزمة java.lang والتي يقوم المترجم باستيرادها أوتوماتيكيا مع كل برنامج جافا دون الحاجة لكتابة جملة استيراد import في بداية البرنامج .
السطر 30

```
total = total + gradeValue ;
```

يقوم بعملية جمع الدرجات وتخزينها في المتغير total
السطر33

```
gradeCounter = gradeCounter + 1;
```

زيادة العداد بمقدار ١ السطر 35
نهاية جملة

```
while        }
```

يتم تكرار السطور من 21 إلى 35 حتى تصبح قيمة gradeCounter أكبر من 10 عند ذلك نقف الحلقة التكرارية وينتقل البرنامج لتنفيذ السطر التالي السطر 38

```
average = total / 10;
```

يقوم هذا السطر بحساب قيمة المتوسط الحسابي وذلك بقسمة مجموع الدرجات total على عددها 10 .

السطور 41 ، 42،

```
43 JOptionPane.showMessageDialog( null, "Class average is " + average,
"Class Average", JOptionPane.INFORMATION_MESSAGE );
```

تمثل جملة الطباعة بواسطة صندوق الحوار .

لاحظ أنه في المثال السابق كان عدد مرات التكرار محدودة ومعروفة مسبقا = 10 وذلك عن طريق الشرط

```
while ( gradeCounter < = 10 )
```

وبذلك حددنا لمستخدم البرنامج أن يقوم بإدخال 10 أرقام .

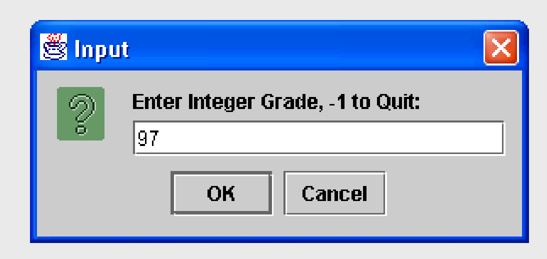
في المثال القادم سوف ترى أنه من الممكن أن تستمر الحلقة التكرارية ويستمر البرنامج في سؤال المستخدم أن يدخل درجة الطالب حتى يقوم المستخدم بإدخال رقم ١ وهو شرط توقف الحلقة

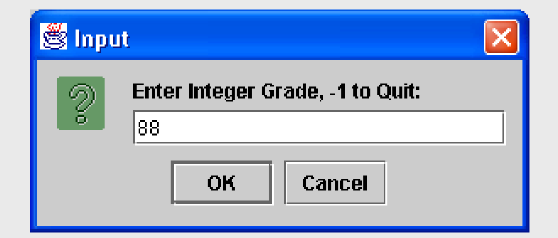
```
while  ( gradeCounter ! = - 1 )
{
//الجمل المراد تكرارها
   }
```

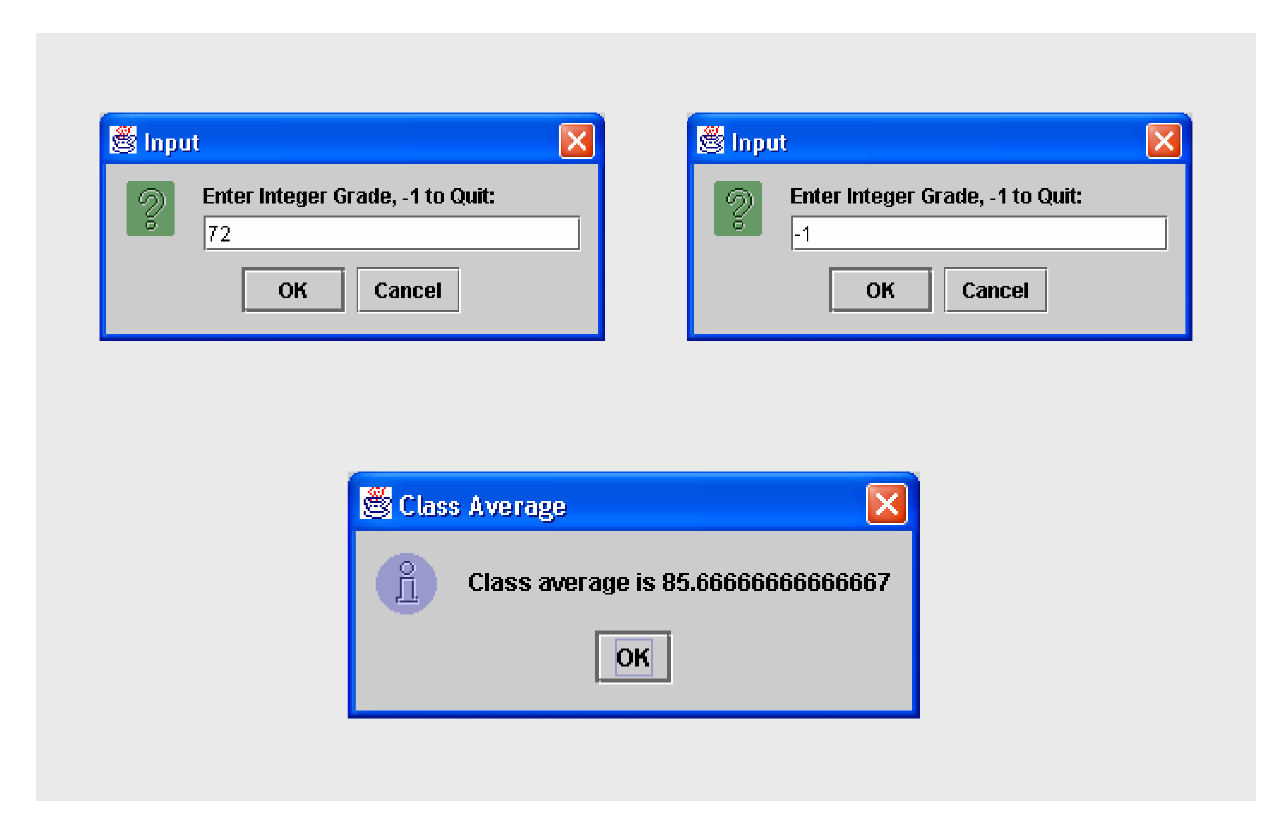
مثال

```
1.  // Fig. 3.7: Average2.java
2.  // Class average program with sentinel-controlled repetition.
3.  // Java extension packages
4.  import javax.swing.JOptionPane;
5.  public class Average2 {
6.  // main method begins execution of Java application
7.  public static void main( String args[] )
8.  {
9.  int gradeCounter,  // number of grades entered
10. gradeValue,    // grade value
11. total;         // sum of grades
12. double average;    // average of all grades
13. String input;      // grade typed by user
```

```
14. // Initialization phase
15. total = 0;          // clear total
16. gradeCounter = 0;  // prepare to loop
17. // Processing phase
18. // prompt for input and read grade from user
19. input = JOptionPane.showInputDialog(
20. "Enter Integer Grade, -1 to Quit:" );
21. gradeValue = Integer.parseInt( input );
22. while ( gradeValue != -1 ) {
23. total = total + gradeValue;
24. gradeCounter = gradeCounter + 1;
25.
26. // prompt for input and read grade from user
27. input = JOptionPane.showInputDialog(
28. "Enter Integer Grade, -1 to Quit:" );
29.
30. // convert grade from a String to an integer
31. gradeValue = Integer.parseInt( input );
32. }
33. if ( gradeCounter != 0 ) {
34. average = (double) total / gradeCounter;
35.
36. // display average of exam grades
37. JOptionPane.showMessageDialog( null,
38. "Class average is " + average,
39. "Class Average", JOptionPane.INFORMATION_MESSAGE );
40. }
41. else
42. JOptionPane.showMessageDialog( null,
43. "No grades were entered", "Class Average",
44. JOptionPane.INFORMATION_MESSAGE );
45. System.exit( 0 );    // terminate application
46. }  // end method main
47. }  // end class Average2
```







شكل (3-7) مثال على استخدام حلقة while

شرح البرنامج

السطور 19 -21

```
input = JOptionPane.showInputDialog(
"Enter Integer Grade, -1 to Quit:" );
gradeValue = Integer.parseInt( input );
```

البرنامج يقرأ قيمة من المستخدم ويقوم بتحويلها إلى القيمة الصحيحة يحتوي البرنامج على حلقة while التكرارية على الشكل

```
while (gradeValue ! = -1 )
{
// الجمل المراد تكرارها
}
```

تتكرر الجمل داخل الحلقة طالما شرط الاستمرار متحقق وهو أن تكون الدرجة لا تساوي ١-

السطر 23 gradeValue ! = -1

total = total + gradeValue ; إذا كان الشرط متحققاً يقوم البرنامج بإضافة الدرجة إلى المتغير total

السطر 24

gradeValue = gradeValue + 1 ; يقوم بزيادة العداد بمقدار الوحدة وذلك لمعرفة عدد الدرجات السطور 27 - 31

```
input = JOptionPane.showInputDialog(
"Enter Integer Grade, -1 to Quit:" );

// convert grade from a String to an integer
gradeValue = Integer.parseInt( input );
```

يتم قراءة درجة أخرى من المستخدم ثم يتم تحويلها إلى القيمة الصحيحة int .

السطر 33–40

```
if ( gradeCounter != 0 ) {
average = (double) total / gradeCounter;
JOptionPane.showMessageDialog( null,
"Class average is " + average,
"Class Average", JOptionPane.INFORMATION_MESSAGE );
}
```

إذا كان المتغير gradeCounter لا يساوى صفراً ومعنى ذلك أن هناك درجات تم قراؤتها أو أن هناك درجة واحدة على الأقل تم قراؤتها ، إذا تحقق هذا الشرط يتم تنفيذ الجملة في السطر التالي وهو ٣٤ وهى لحساب المتوسط الحسابي ، لاحظ أننا استخدمنا الأمر (double) وذلك لأن كلاً من المتغير gradeCounter , total معرفين على أنهما صحيحان int وبما أن المتوسط الحسابي يمكن أن يكون قيمة ذات كسور عشرية لذلك استخدمنا الأمر (double) .ثم بعد ذلك يتم طباعة قيمة المتوسط الحسابي في صندوق حوار .

السطور 42 - 44

```
JOptionPane.showMessageDialog (null ,” no grades were entered “
(“ class average “ , JOptionPane.INFORMATION_MESSAGE ) ;
```

هي جملة طباعة يتم تنفيذها في حالة عدم إدخال أي درجة

السطر 45

```
System.exit ( 0 ) ;
```

يتم إضافة هذه الجملة عند استعمال الفصل JOptionPane

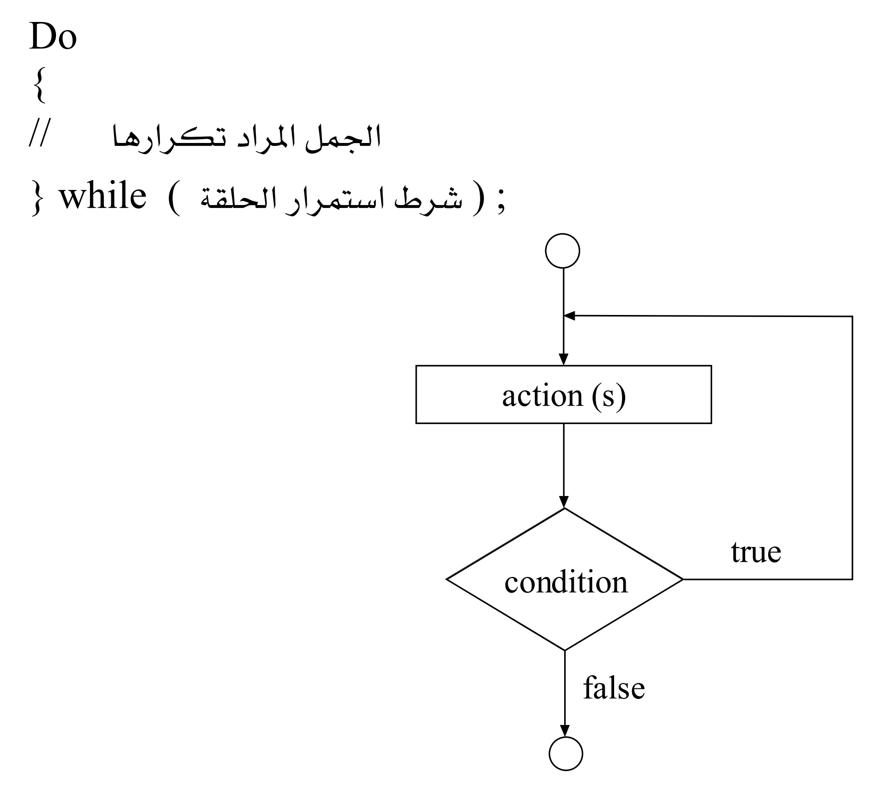
خطأ شائع:

عدم كتابة الأقواس بعد جملة while وذلك إذا كان المطلوب تكرار أكثر من جملة يعطى خطأ منطقي وذلك لأنه إذا لم يوجد أقواس يتم تكرار الجملة التالية لحلقة while مباشرة فقط .

حلقة **do/while** التكرارية

تستخدم حلقة do–while كسابقتها while لعمل تكرار لجملة أو عدة جمل ويكون التركيب البنائي لها على الشكل :

```
Do
{
//      الجمل المراد تكرارها
} while ( شرط استمرار الحلقة ) ;
```



شكل (3-8) خريطة تدفق حلقة do/while

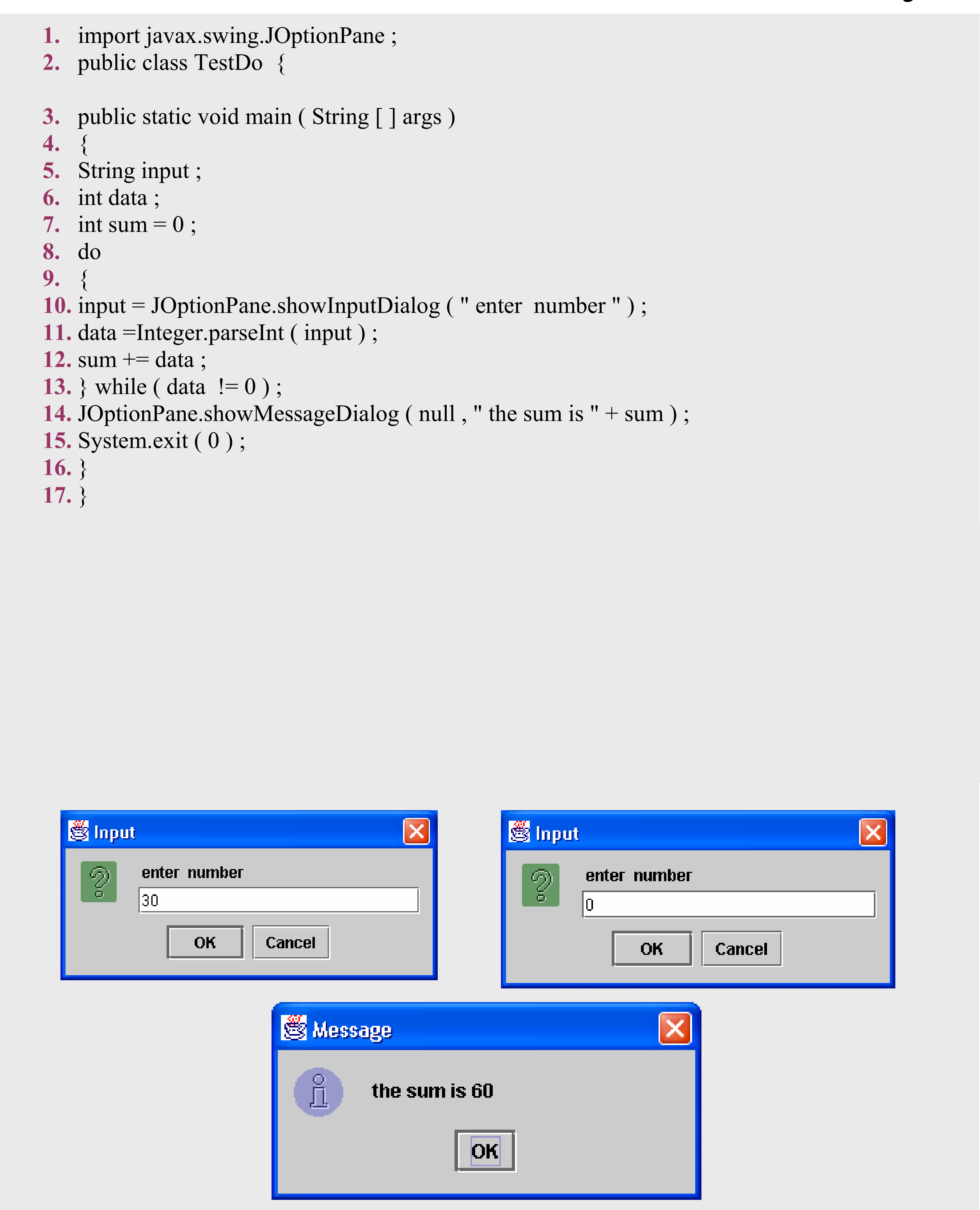
لاحظ أن الجمل المراد تكرارها تنفذ مرة واحدة على الأقل قبل أن يتم اختبار شرط استمرار الحلقة والذي يكون بداخل الأقواس بعد while فإذا كان الشرط صحيحاً true يتم التكرار والعودة لتنفيذ الجمل أما إذا كان خطأ false تتوقف الحلقة فورا . لذلك فإننا نرى الفرق بين جملة while وجملة do /while وهو أن الجمل يتم تنفيذها مرة واحدة على الأقل حتى لو كان شرط استمرار الحلقة خطأ false وذلك على عكس while التي تختبر الشرط أولا فإذا كان صحيحا يتم التنفيذ والتكرار وإذا كان خطأ تتوقف فورا دون تنفيذ الجمل داخل الحلقة .

المثال التالي يوضح فكرة عمل الحلقة do/while

مثال

```
1.  import javax.swing.JOptionPane ;
2.  public class TestDo  {

3.  public static void main ( String [ ] args )
4.  {
5.  String input ;
6.  int data ;
7.  int sum = 0 ;
8.  do
9.  {
10. input = JOptionPane.showInputDialog ( " enter  number " ) ;
11. data =Integer.parseInt ( input ) ;
12. sum += data ;
13. } while ( data  != 0 ) ;
14. JOptionPane.showMessageDialog ( null , " the sum is " + sum ) ;
15. System.exit ( 0 ) ;
16. }
17. }
```



شكل (3-9) مثال على استخدام حلقة do/while

هذا البرنامج يقوم بقراءة أرقام من المستخدم عن طريق توجيه رسالة له في صندوق حوار ثم إذا أدخل المستخدم رقم صفر يقوم البرنامج بطباعة حاصل جمع الأرقام المدخلة في صندوق رسالة .

لاحظ استخدام جملة do /while في السطور من ٨ إلى ١٣

داخل الحلقة تم توجيه رسالة للمستخدم ليدخل رقماً أو يدخل صفراً عند الانتهاء

السطر رقم ١١

```
data =Integer.parseInt ( input ) ;
```

تحويل القيمة من النوع سلسلة String إلى النوع الصحيح int

السطر رقم ١٢

```
sum += data ;
```

هو عبارة عن عملية الجمع ويتم إضافة الرقم الصحيح إلى المتغير sum

السطر رقم ١٣

```
} while ( data ! = 0 ) ;
```

هو عبارة عن إغلاق القوس { ثم يأتي بعد ذلك اختبار شرط استمرار الحلقة وهو هل قيمة الرقم المدخل لا تساوي صفراً فإذا كان الجواب بنعم يتم إعادة تكرار الحلقة وإن كان الجواب بلا فتتوقف الحلقة فورا وننتقل إلى السطر التالي.

السطر رقم ١٤

```
JOptionPane.showMessageDialog ( null , “ the sum is “ + sum ) ;
```

هو جملة طباعة في صندوق رسالة يتم فيها طباعة قيمة المتغير sum وهى حاصل جمع الأعداد المدخلة من قبل المستخدم

السطر رقم ١٥

```
System.exit ( 0 )
```

كما قلنا سابقا هذا الأمر يكتب في النهاية عند استخدام الفصل JoptionPane

ملحوظة : في هذا المثال أيضا كان عدد مرات تكرار الحلقة غير معلوم .

حلقة for التكرارية

تستخدم أيضا هذه الجملة لعمل تكرار لجملة أو عدة جمل ويكون التركيب البنائى لها على الشكل :

for (إعطاء قيمة ابتدائية للعداد ; شرط استمرار الحلقة ; جملة زيادة أو نقصان العداد)
{
//الجملة المراد تكرارها
}

i = 0.01f

i <= 1.0f

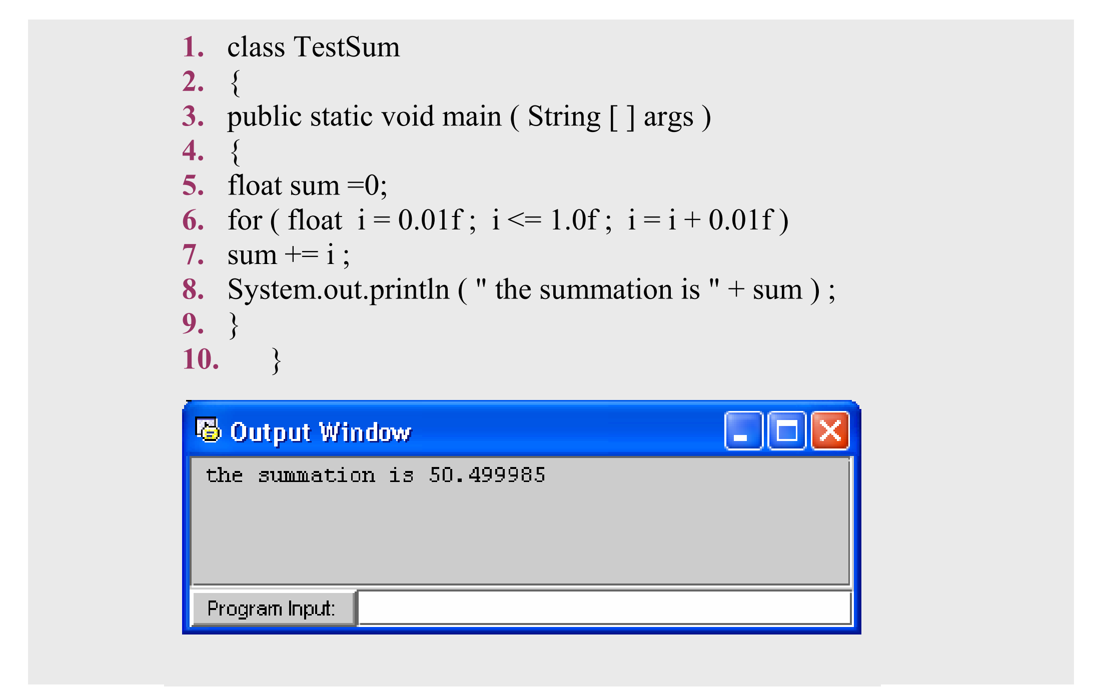
true

sum = sum + i

i = i+1

false

شكل (3-10) خريطة تدفق حلقة for



```
1.  class TestSum
2.  {
3.  public static void main ( String [ ] args )
4.  {
5.  float sum =0;
6.  for ( float  i = 0.01f ;  i <= 1.0f ;  i = i + 0.01f )
7.  sum += i ;
8.  System.out.println ( " the summation is " + sum ) ;
9.  }
10.     }
```


شكل (3-11) مثال على استخدام حلقة for

شرح البرنامج

هذا البرنامج يقوم بجمع الأعداد العشرية من 0.01 وحتى 1.0

السطر رقم 5

```
float sum =0;
```

تعريف المتغير sum من النوع float أي يقبل الكسور العشرية و تم إعطاؤه قيمة ابتدائية صفر

السطر رقم 6

```
for ( float  i = 0.01f ;  i <= 1.0f ;  i = i + 0.01f )
```

جملة for التكرارية وتتكون من ٣ أجزاء :

الجزء الأول

```
float i = 0.01f
```

تعريف العداد وإعطاؤه قيمة ابتدائية تساوي 0.01

الجزء الثاني

```
i < = 1.0f
```

شرط استمرار الحلقة وهو أن يكون العداد i أقل من أو يساوي 1.0 بمعنى أنة إذا زادت قيمة العداد عن 1.0 تتوقف الحلقة فورا .

الجزء الثالث

```
i = i + 0.01f
```

زيادة العداد i بمقدار 0.01

السطر رقم ٧

```
sum += i ;
```

عملية الجمع وتتم بإضافة قيمة العداد i في كل مرة إلى المتغير sum

السطر رقم ٨

```
System.out.println ( " the sum is " + sum ) ;
```

يمثل جملة الطباعة التي تقوم بطباعة المتغير sum وهو عبارة عن حاصل جمع الأرقام

0.01 + 0.02 + 0.03 + ----------- + 0.1

أمثلة على استخدام جملة for

١ - تغير العداد من 1 إلى 100 بزيادة العداد في كل حلقة بمقدار 1

= 100 ; i + +)<for (int f = 1 ; i

٢ - تغير العداد من 7 إلى 77 بزيادة العداد في كل مرة بمقدار 7

= 77 ; i + = 7)<for (int i = 1 ; i

٣ - تغير العداد من 20 إلى 2 بإنقاص العداد في كل مرة بمقدار 2

= 2 ; i - = 2)>for (int i = 20 ; i

٤ - تغير العداد بالقيم التالية على الترتيب 2 , 5 , 8 , 11, 14 , 17 , 20

= 20 ; i + = 3) <for (int i= 2 ; i

٥ - تغير العداد بالقيم التالية على الترتيب 99 , 88 , 77 , 66 , 55 , 44 , 33 , 22 , 11

= 0 ; 0 - = 11)>for (int j = 99 ; j

خطأ شائع

وضع فاصلة فقط بدلا من الفاصلة المنقوطة التي تفصل بين أدوات التحكم في جملة for يعطي خطأ في بناء الجملة syntax error .

حلقات for المتداخلة

المثال التالي يستخدم الحلقات المتداخلة لطباعة جدول الضرب ، تتكون الحلقات المتداخلة من حلقة خارجية وحلقة أخرى داخلية أو أكثر ، وفي كل مرة تتكرر الحلقة الخارجية يتم تكرار الحلقات الداخلية من بداية العداد إلى نهايته.

مثال

```
1.  class TestMultable
2.  {
3.  public static void main ( string [ ]  args )
4.  {
5.  // display the title
6.  System.out.println ( "          multiplication table " ) ;
7.  System.out.println ( " --------------------------------------------- " );
8.  // display the number title
9.
10. System.out.print ( "  | " ) ;
11. for ( int   j=1 ;  j <= 9;  j ++ )
12. System.out.print ( "   " + j ) ;
13. System.out.println ( " " ) ;
14.
15. // print table body
16. for ( int  i = 1 ;  i <= 9 ;  i ++ )
17. {
18. System.out.print ( i + " | " ) ;
19. for ( int  j = 1 ;  j <= 9 ;  j ++ )
20. {
21.
22. // display the product and align properly
23.
24. if ( i * j  < 10 )
25. System.out.print ( "   " + i* j ) ;
26. else
27. System.out.print ( " " + i * j ) ;
28. }
29. System.out.println ( " " ) ;
30. }
31. }
32. }
```

```
Output Window
             multiplication table
 ---------------------------------------------
  |    1   2   3   4   5   6   7   8   9
1 |    1   2   3   4   5   6   7   8   9
2 |    2   4   6   8  10  12  14  16  18
3 |    3   6   9  12  15  18  21  24  27
4 |    4   8  12  16  20  24  28  32  36
5 |    5  10  15  20  25  30  35  40  45
6 |    6  12  18  24  30  36  42  48  54
7 |    7  14  21  28  35  42  49  56  63
8 |    8  16  24  32  40  48  56  64  72
9 |    9  18  27  36  45  54  63  72  81
Program Input:
```

شكل (3-12) مثال على الحلقات المتداخلة

شرح البرنامج

السطر رقم ٦

```
System.out.println ( "     multiplication table " ) ;
```

يقوم بطباعة العنوان multiplication table

السطر رقم ٧

```
System.out.println ( " --------------------------------------------- " );
```

يقوم بطباعة السطر الثاني وهو عبارة عن فاصل - - - - - - - - - - - - - - - - - - -

السطور ١١ إلى ١٢

```
for ( int   j=1 ;  j <= 9;  j ++ )
System.out.print ( "  " + j ) ;
```

يقوم بطباعة الأرقام من 1 إلى 9 وذلك في السطر الثالث

السطور ١٦ إلى ٣٠

عبارة على حلقة for متداخلة ، الحلقة الخارجية لها عداد i والحلقة الداخلية لها عداد j .

لاحظ أنه لكل i جديدة يعرض حاصل ضرب j*i ويتم ذلك في الحلقة الداخلية حيث قيم j تتراوح من 1 إلى 9

السطر رقم ٢٤

```
if ( i*j   <  10 )
```

عبارة عن جملة if الشرطية وهى تبحث ما إذا كان حاصل ضرب i و j أقل من 10 أم لا وهذا الشرط يفيد في عمل محاذاة للأرقام أثناء الطباعة ، فإذا تحقق الشرط نفذ البرنامج للسطر التالي مباشرة وهو

```
System.out.print ( "  " + i*j ) ;
```

وهذا السطر يقوم بطباعة مسافتين قبل طباعة حاصل ضرب أل i*j .

أما إذا لم يتحقق الشرط وكان حاصل الضرب i*j أكبر من أو يساوي 10

معنى ذلك أن العدد الناتج يأخذ خانتين في الطباعة لذلك تنفذ جملة else

```
System.out.print ( " " + i*j ) ;
```

لاحظ طباعة مسافة واحدة قبل طباعة حاصل الضرب

السطر رقم ٢٩

```
System.out.println ( " " ) ;
```

لطباعة سطر خالٍ في نهاية كل سطر

جمل continue **و** break

تستعمل هذه الجمل عندما يراد تغيير المسار الطبيعي للبرنامج فمثلا عندما تستخدم جملة break داخل بناء جملة while , for , do/while , switch تسبب الخروج منها فورا ويستمر تنفيذ باقي جمل البرنامج التي تلي بناء الجملة والاستخدام الشائع لجملة break هو للهروب مبكرا من تنفيذ حلقة أو لإهمال تنفيذ باقي جملة switch .

وفي المثال التالي سوف نقوم بتوضيح عمل جملة break

مثال

```
1.   // Fig. 3.13: BreakTest.java
2.   // Using the break statement in a for structure
3.
4.   // Java extension packages
5.   import javax.swing.JOptionPane;
6.
7.   public class BreakTest {
8. // main method begins execution of Java application
9. public static void main( String args[] )
10. {
11. String output = "";
12. int count;
13.
14. // loop 10 times
15. for ( count = 1; count <= 10; count++ ) {
16.
17. // if count is 5, terminate loop
18. if ( count == 5 )
19. break;  // break loop only if count == 5
20.
21. output += count + " ";
22.
23. }  // end for structure
24.
25. output += "\nBroke out of loop at count = " + count;
26. JOptionPane.showMessageDialog( null, output );
27.
28. System.exit( 0 );  // terminate application
29.
30. }  // end method main
31.
32. }  // end class BreakTest
```

Message

1 2 3 4

Broke out of loop at count = 5

OK

شكل (3-13) مثال على استخدام جملة break

شرح البرنامج :

في البرنامج السابق لاحظ وجود جملة if في السطر رقم 18 وهى موجودة داخل بناء جملة for وشرط جملة if أن يكون المتغير count يساوي 5 فإذا تحقق هذا الشرط نفذت الجملة التالية رقم 19 وفيها نرى جملة ; break ، هذه الجملة تتسبب في إنهاء الحلقة التكرارية والخروج منها لينفذ البرنامج بعدها مباشرة أول جملة بعد الحلقة وهى في السطر رقم 25

```
output + =  " in broke out of  loop at count = " + count ;
```

لاحظ إضافة النص السابق إلى المتغير output بالإضافة إلى قيمة المتغير count وذلك قبل طباعته باستخدام صندوق رسالة في السطر رقم 26

```
JOptionPane.showMessageDialog ( null , output ) ;
```

وفي المثال التالي سوف نقوم بتوضيح عمل جملة continue ولنرى تأثير جملة continue والفرق بينها وبين جملة break نقوم بكتابة نفس البرنامج السابق ولكن نستبدل جملة break بجملة continue نلاحظ أن جملة continue تتعدى الجملة الباقية في الحلقة لتبدأ تنفيذ الحلقة من البداية بالقيمة التالية للعداد .

ففي المثال إذا تحقق الشرط وكانت قيمة count تساوي 5 قام البرنامج بتنفيذ جملة continue والتي تجعل البرنامج يهمل باقي الجمل داخل الحلقة وهى في المثال السطر رقم 29

```
Output + = count + " " ;
```

ثم يعود لتنفيذ الحلقة من البداية بالقيمة التالية للعداد count وهى 6

ويكون شكل تنفيذ البرنامج باستخدام جملة continue بدلا من جملة break على الشكل التالي

مثال

```
1.  // Fig. 3.14: ContinueTest.java
2.  // Using the continue statement in a for structure
3.  // Java extension packages
4.  import javax.swing.JOptionPane;
5.
6.  public class ContinueTest {
7.
8.  // main method begins execution of Java application
9.  public static void main( String args[] )
10. {
11. String output = "";
12.
13. // loop 10 times
14. for ( int count = 1; count <= 10; count++ ) {
15. // if count is 5, continue with next iteration of loop
16. if ( count == 5 )
17. continue;  // skip remaining code in loop
18. // only if count == 5
19. output += count + " ";
20. }  // end for structure
21. output += "\nUsed continue to skip printing 5";
22. JOptionPane.showMessageDialog( null, output );

23. System.exit( 0 );  // terminate application

24. } // end method main

25. }  // end class ContinueTest
```

Message

1 2 3 4 6 7 8 9 10
Used continue to skip printing 5

OK

شكل (3-14) مثال على استخدام جملة continue

جمل **break** و **continue** المعنونة

تعرضنا سابقا لجملة break وقلنا أنها تتسبب في الهروب من بناء الجملة التي تشتمل عليها فقط سواء كان هذا البناء هو جملة while أو for أو do/while أو switch

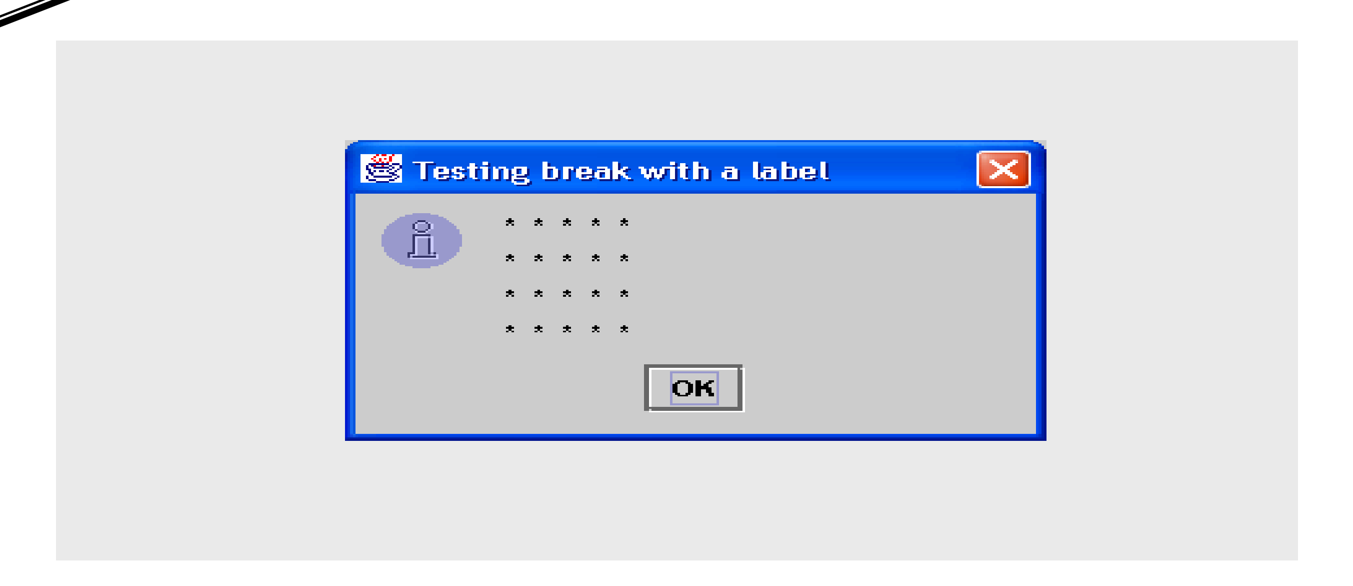
ولكي نستطيع عمل هروب من مجموعة من بناءات الجمل معا لابد أن نستخدم جملة الهروب المعنونة labeled break ، عندما تستخدم هذه الجملة وتنفذ داخل بناء while أو for أو do/while أو switch تتسبب في الهروب فورا من هذا البناء وأي عدد آخر من البناءات التي تشتمل عليها ، ثم بعد ذلك يستأنف تنفيذ البرنامج بعد القالب المعنون (القالب المعنون هو عبارة عن مجموعة من الجمل داخل البرنامج والتي تكون محصورة بين قوسين وفي بدايتها عنوان)

```
Stop : {
            // جمل القالب
       }
```

تستخدم عادة جملة الهروب المعنونة labeled break للخروج من الحلقات المتداخلة والتي يمكن أن تكون بواسطة while أو for أو do/while أو switch

مثال

```
1. // Fig. 3.15: BreakLabelTest.java
2. // Using the break statement with a label
3.
4. // Java extension packages
5. import javax.swing.JOptionPane;
6.
7. public class BreakLabelTest {
8.
9. // main method begins execution of Java application
10. public static void main( String args[] )
11. {
12. String output = "";
13.
14. stop: {   // labeled block
15.
16. // count 10 rows
17. for ( int row = 1; row <= 10; row++ ) {
18. // count 5 columns
19. for ( int column = 1; column <= 5 ; column++ ) {
20.
21. // if row is 5, jump to end of "stop" block
22. if ( row == 5 )
23. break stop; // jump to end of stop block
24.
25. output += "*  ";
26.
27. }  // end inner for structure
28.
29. output += "\n";
30.
31. }  // end outer for structure
32.
33. // the following line is skipped
34. output += "\nLoops terminated normally";
35.
36. }  // end labeled block
37.
38. JOptionPane.showMessageDialog(
39. null, output,"Testing break with a label",
40. JOptionPane.INFORMATION_MESSAGE );
41.
42. System.exit( 0 );  // terminate application
43.
44. }  // end method main
45.
46. }  // end class BreakLabelTest
```



شكل (3-15) مثال على استخدام جملة break المعنونة

شرح البرنامج:

هذا المثال يوضح استخدام جملة الهروب المعنونة مع الحلقات المتداخلة نلاحظ من البرنامج أن القالب في السطور من 14 إلى 36 كما أنه يبدأ بعنوان (دائما ما يكون العنوان عبارة عن معرف متبوع بـ (:

سطر 14

```
stop: {   // labeled block
```

بداية القالب واسم العنوان

السطور ١٧ - ٣١

عبارة عن حلقات for المتداخلة

السطر ٣٤

```
output += "\nLoops terminated normally";
```

إضافة النصLoop terminated normally إلى المتغير output.

نلاحظ أنه عندما تكون قيمة المتغير row تساوي 5 أي يتحقق الشرط الموجود في السطر رقم ٢٢ يتم

تنفيذ جملة الهروب المعنونة الموجودة في السطر رقم ٢٣

```
if ( row == 5 )
break stop;  // jump to end of stop block
```

هذه الجملة تنهي عمل كل من بناء for الموجودة في السطر رقم ١٩ وبناء جملة for الخارجية والموجودة في السطر رقم ١٧ ويستأنف تنفيذ بقية البرنامج بداية من السطر رقم ٣٨ أي أول سطر بعد القالب المعنون.

ملحوظة: بناء جملة for الخارجية يتم تنفيذ ما بها من جمل 4 مرات فقط (حتى تصل قيمة المتغير row إلى 5) لذلك السطر رقم ٣٤ لا ينفذ أبدا وذلك لأنه داخل القالب المعنون وجملة for الخارجية لا تكتمل أبدا .

كما قلنا سابقا فإن جملة continue المعنونة تهمل تنفيذ باقي الجمل في الحلقة لتبدأ تنفيذ الحلقة من البداية بالقيمة التالية للعداد أما جملة continue المعنونة labeled continue فهي تتسبب في إهمال باقي الجمل في الحلقة وأي حلقات أخرى تشتمل عليها ، ثم تبدأ بتنفيذ بناء التكرار المعنون structure labeled repetition الذي يشملها وذلك بقيمة جديدة للعداد في كل الحلقات.

بناء التكرار المعنون هو حلقة تكرارية تبدأ بعنوان.

مثال

```
1.  // Fig. 3.16: ContinueLabelTest.java
2.  // Using the continue statement with a label
3.
4.  // Java extension packages
5.  import javax.swing.JOptionPane;
6.
7.  public class ContinueLabelTest {
8.
9.  // main method begins execution of Java application
10. public static void main( String args[] )
11. {
12. String output = "";
13.
14. nextRow:   // target label of continue statement
15.
16. // count 5 rows
17. for ( int row = 1; row <= 5; row++ ) {
18. output += "\n";
19.
20. // count 10 columns per row
21. for ( int column = 1; column <= 10; column++ ) {
22.
23. // if column greater than row, start next row
24. if ( column > row )
```

```
25. continue nextRow; // next iteration of
26.
27. // labeled loop
28.
29. output += "*  ";
30.
31. }  // end inner for structure
32.
33. }  // end outer for structure
34.
35. JOptionPane.showMessageDialog(
36. null, output,"Testing continue with a label",
37. JOptionPane.INFORMATION_MESSAGE );
38.
39. System.exit( 0 );  // terminate application
40.
41. }  // end method main
42.
43. }  // end class ContinueLabelTest
```

```
Testing continue with a label
*
* *
* * *
* * * *
* * * * *
OK
```

شكل (3-16) مثال على استخدام جملة continueالمعنونة

أسئلة وتمارين على التحكم البنائي

١ ضع علامة صح أمام العبارة الصحيحة وعلامة خطأ أمام العبارة الخاطئة لكل من الجمل التالية

أ - لابد من وجود الحالة الافتراضية default داخل بناء switch

ب – لابد من وجود جملة break بعد الحالة الافتراضية default داخل بناء switch

ج _ التعبير (x > y && a < b) صحيحاً إذا كان x > y صحيحاً أو a < b صحيح

د _ يقال التعبير يحتوي على العامل ١١ انه صحيح إذا كان أحد المعاملات صحيحاً أو كلاهما معا .

٢ اكتب جملة أو عدة جمل بلغة الجافا لكي تقوم بعمل المهام التالية

أ _ جمع الأعداد الفردية من 1 إلى 99 باستخدام حلقة for ، افرض أن المتغيرات الصحيحة sum Count قد تم تعريفها .

ب _ طباعة الأعداد الصحيحة من 1 إلى 20 باستخدام حلقة while

افرض أن متغير العدد هو x قد تم تعريفه ولكن لم يعطَ له القيمة الابتدائية

اطبع 5 أعداد فقط في كل سطر

ملحوظة : استخدم x % 5 إذا كان ناتج التعبير السابق يساوي ٥

اطبع سطراً جديداً وإذا لم يكن يساوي صفراً يتم طباعة مسافة فقط

ج _ كرر السؤال السابق ولكن باستخدام حلقة for

٣ - وضح ناتج تنفيذ هذا البرنامج ؟

```
1  public class printing  {
2
3           public static void main ( String args [ ] )
4            {
5               for ( int  i = 1; i  <= 10;  i++ ) {
6
7                   for ( int  j = 1; j  <= 5;  j++ )
8                       System.out.print ('@' );
9
10                  System.out.println ( );
11
12             }
13
14         }
15
16    }
```

4 - ما العمل الذي يقوم به هذا الجزء من البرنامج ؟

```
        for  (  i = 1;  i  <= 5;  i++ ) {

             for  (  j = 1;  j  <= 3;  j++ ) {

                  for  (  k = 1;  k  <= 4;  k++ ) {
                       System.out.print ( ' * ' );

                 System.out.println ( );
        }

        System.out.println ( );
   }
```

٥ - اكتب برنامجاً يوجد الرقم الأصغر لمجموعة من الأرقام الصحيحة المُدخلة بواسطة المُستخدم، افرض أن الرقم الأول يمثل عدد الأرقام.

٦ - اكتب برنامج يقوم بحساب حاصل ضرب الأعداد الفردية من 1 إلى 15، ثم يقوم بعرض الناتج في صندوق رسالة.

٧ - يستخدم المضروب في كثير من المسائل الرياضية، ومضروب العدد 8 (يكتب بالشكل !8 ويُقال له مضروب 8) والمضروب هو عبارة عن حاصل ضرب الأعداد الصحيحة الموجبة من 1إلى 8.
اكتب برنامجاً يقوم بحساب مضروب الأعداد الصحيحة من 1 إلى 5، واعرض الناتج داخل صندوق رسالة.

٨ - باستخدام الحلقات المُتداخلة، اكتب برنامجاً يقوم بعرض كلٍ من الأشكال التالية:

- ملحوظة: برنامج لكل شكل

```
(a)              (b)                 (c)               (d)

         *     **********  **********       *
        **      *********  *********        **
       ***       ********  ********         ***
      ****        *******  *******          ****
     *****         ******  ******           *****
    ******          *****  *****            ******
   *******           ****  ****             *******
  ********            ***  ***              ********
 *********             **  **               *********
**********              *  *                **********
```

٩ - اكتب برنامج يقوم بحساب مجموع المتوالية غير المنتهية

$$\pi = 4 - \frac{4}{7} - \frac{4}{5} + \frac{4}{3} + \frac{4}{9} - \frac{4}{11} + ...$$

اطبع جدولاً به قيم π مُقربة باستخدام حد واحد من المتوالية السابقة، ثم باستخدام حدين من المتوالية، ثم باستخدام ثلاثة حدود، ثم أوجد عدد الحدود المستخدمة لتكون π تساوي 3.14159

١٠- وضح ناتج تنفيذ هذا البرنامج ؟

```
1  public class Mystery2  {
2
3           public static void main ( string args [ ] )
4            {
5               int count  = 1;
6
7              while  (  count <= 10  )  {
8                      System.out.println (
9                             count %  2 = = 1 ? "*****"   :  "+++++++++" );
10                      ++count;
11                 }
12           }
13     }
```

١١- وضح ناتج تنفيذ هذا البرنامج ؟

```
1  public class Mystery3  {
2
3           public static void main ( String args [ ] )
4            {
5               int row  = 10,  column;
6
7              while  (  row >= 1  )  {
8                  column = 1;
9
10                 while  (  column <= 10  )  {
11                     System.out.print (row %  2 = = 1 ? "<"  :  ">" );
12                     ++column;
13                  }
14
15                --row;
16                System.out.println( );
17            }
18        }
19     }
```

ملحـــــق أ

لكي نقوم بكتابة برنامج بلغة الجافا ثم تنفيذه لابد لنا من وجود :

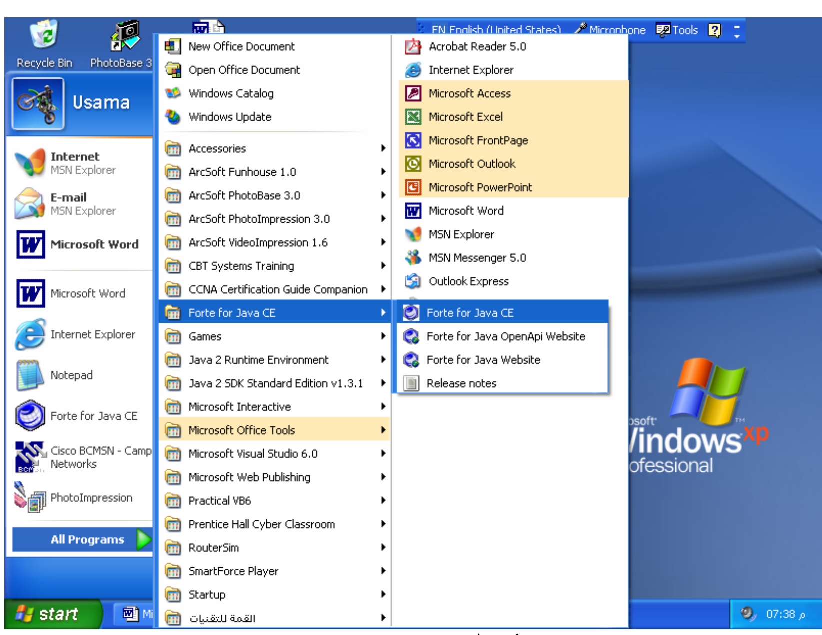
أولا : Java 2 Software Development Kit والمعروفة اختصارا بـ JDK وموجود منها الآن الإصدار رقم 1.4 ، وهي عبارة عن تعليمات اللغة نفسها.

ثانيا : Integrated Development Environment والمعروفة اختصارا بـ IDE وهي عبارة عن البيئة التي نكتب فيه البرنامج أو المحرر .

برنامج الـ Forte هو أحد البرامج التي أنتجتها وطورتها شركة صن مايكروسيستمز Sun Microsystems لكي يستخدمه مبرمجو لغة الجافا في تطوير البرامج (تصميم وكتابة وترجمة ثم تنفيذ) أي هو عبارة عن IDE ، لذلك لابد قبل تحميل هذا البرنامج أن نحمل الـ JDK ثم بعد ذلك نقوم بتحميل برنامج الـ Forte ، وأثناء عملية التحميل يطلب منا أن نحدد مسار الـ JDK .

وسوف نتعرض الآن لكيفية كتابة برنامج بسيط بلغة الجافا بواسطة برنامج Forte ومن ثم عمل ترجمة له ثم تنفيذه

١ - تشغيل برنامج الــForte



شكل (A-1) طريقة تشغيل البرنامج

٢ - اختيار **New file**

عند بدء التشغيل تظهر الشاشة المقابلة شكل (A-2) فنختار منها New كما في الشكل وذلك لإنشاء ملف جديد.



شكل (A-2)

٢ - اختيار النموذج **Template**

تظهر بعد ذلك شكل(A-3) المقابل لنختار منها نوع البرنامج المطلوب عمله، وتفيد هذه الطريقة في أن البرنامج المختار يتم فتحه وكتابة الأشياء الأساسية

به مثل تعريف الفصل ، بداية ونهاية البرنامج ، ... الخ

على سبيل المثال نختار Classes ثم من القائمة المسدلة نختارMain إذا كنا نريد عمل برنامج تطبيق .

في خانة Name نكتب اسم الفصل

New... - Template Chooser
Select a template
Templates
AWT Forms
Beans
Classes
Sample Forms
JSP
Other
Swing Forms
Template Description
Contains all templates available in the IDE.
Each template lets you create a new kind of file.
New...
Name: <default name>
Package:
Browse...
Directory: C:\
< Previous | Next > | Finish | Cancel | Help

شكل (A-3)

نلاحظ وجود بعض الاختيارات الأخرى إذا ضغطنا على زر Next لن نتعرض اليها الآن ولكننا سنضغط زر Finish مباشرة وتظهر الشاشة التالية شكل (A-4) التي تسأل المستخدم إذا كان يريد إضافة هذا الفصل إلى المشروع لكي يظهر الفصل في مستكشف المشروع وتكون إجابتنا بنعم Yes

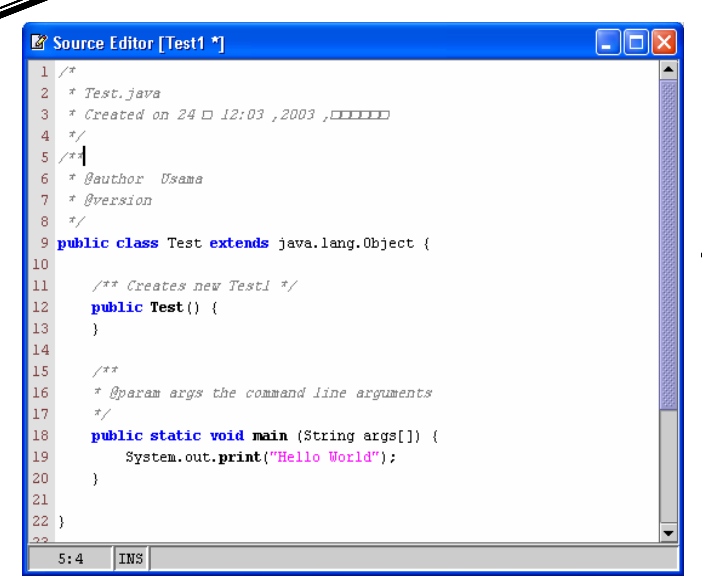
Question
Object Test is not in the current project, so it is not accessible from the Project tab in the Explorer. Would you like to add it to the project?
Yes | No | Always | Never

شكل (A-4)

٣ - كتابة البرنامج

تظهر لنا الشاشة المقابلة شكل (A-5) لنقوم بكتابة البرنامج ونلاحظ وجود تعريف الفصل بالاسم الذي اخترناه من قبل بالإضافة لوجود الطريقة Main التي لابد من كتابتها إذا كنا نقوم بعمل تطبيق.

نقوم بكتابة البرنامج داخل الطريقةMain



شكل (A-5)

٤ - عمل ترجمة Compilation

بعد كتابة البرنامج يتم عمل ترجمة له كما بالشكل(A-6) من قائمة Build نختار Compile أو F9 أو من شريط الأدوات كما بالشكل يتم عمل ترجمة للبرنامج فإذا كانت هناك أخطاء نقرأ الجملة التالية بجانب شريط الأدوات

شكل (A-6)

Error Compiling Test حيث Test هو اسم الفصل وتظهر لنا شاشة الخرج وبها تعريف الخطأ ورقم السطر الموجود به بالإضافة إلى تحديد السطر داخل البرنامج بلون مختلف. شكل (A-6)

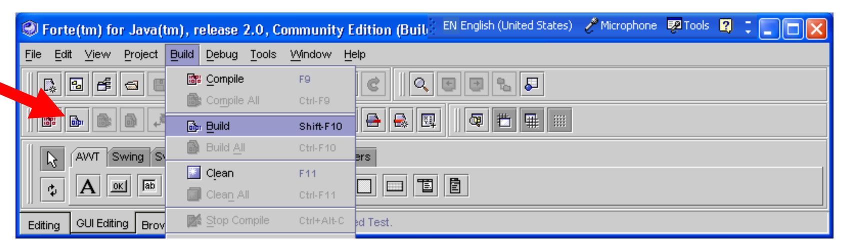
لاحظ أن الخطأ في المثال هو نسيان الفاصلة المنقوطة وحدد الخطأ في السطر رقم ٢٤ وأن السطر ٢٤ هو السطر التالي للسطر الذي وجد به الخطأ لاحظ أيضا تحديد السطر باللون الأخضر في شاشة البرنامج المصدر.

نقوم بعد ذلك بتصحيح الخطأ ثم نعمل ترجمة مرة أخرى هذه المر تظهر الجملة التالية بجوار

شكل (A-6)

شريط الأدوات Finished Test وهي تعني أن البرنامج تم عمل له ترجمة وتم إنشاء ملف بنفس الاسم ولكن بامتداد class.

بعد ذلك نقوم بعمل Build للبرنامج من قائمة Build نختار Build أو نضغط Shift+f10 أو من شريط الأدوات كما بالشكل(A-7)



شكل (A-7)

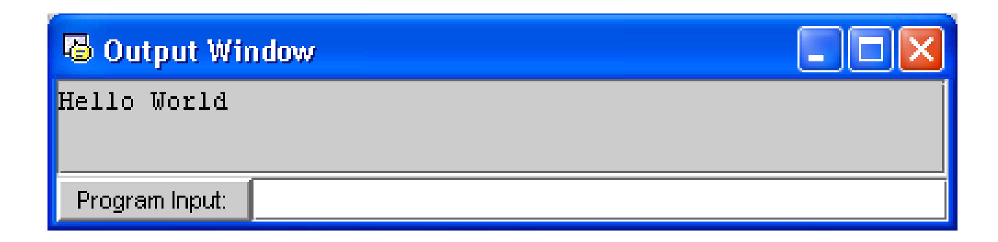
٥ - التنفيذ

يتم عمل تنفيذ للبرنامج من خلال اختيار Execute من قائمة Build أو F6 أو من خلال شريط الأدوات



شكل(A-8)

يظهر ناتج تنفيذ البرنامج في شاشة الخرج Output Window كما بالشكل (A-9)

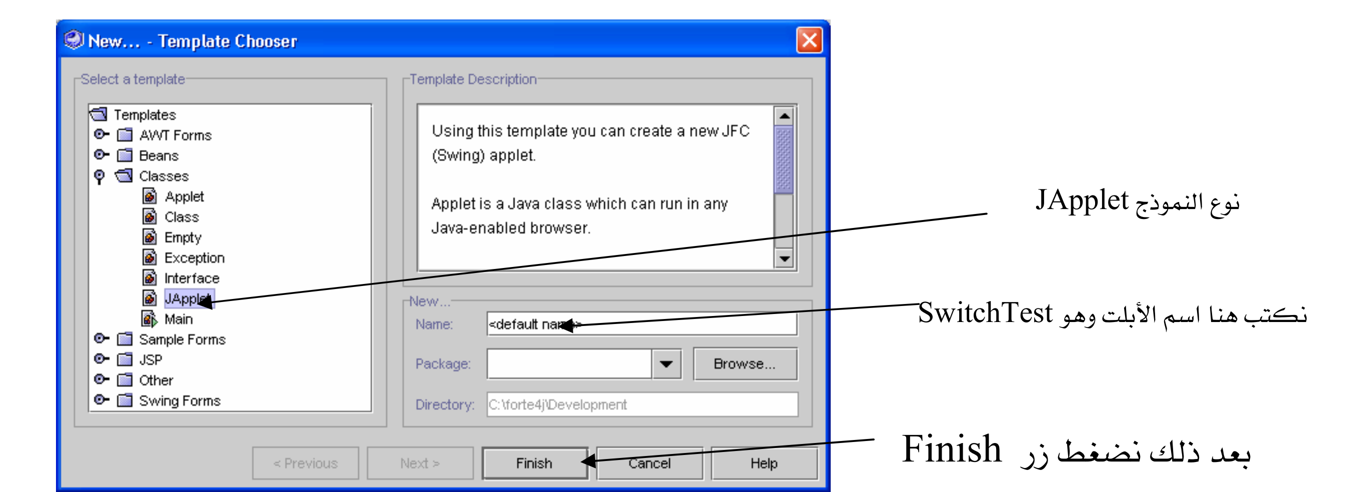


شكل(A-9)

مثال

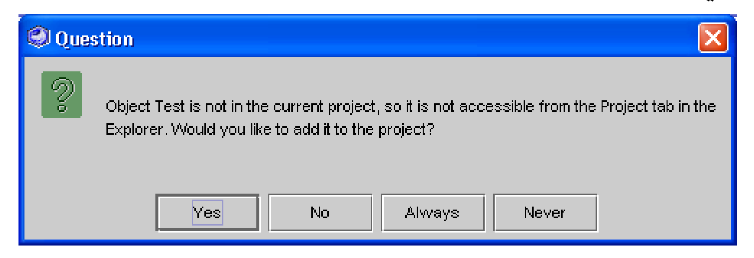
في هذا المثال نقوم باستخدام نموذج آخر لكتابة البرنامج وهو النموذج JApplet الموجود تحت القائمة Classes

اختيار النموذج انظر الشكل(A-10)



شكل (A-10)

وتظهر الشاشة التالية شكل(A-11) التي تسأل المستخدم اذا كان يريد إضافة هذا الفصل إلى المشروع لكي يظهر الفصل في مستكشف المشروع وتكون إجابتنا بنعم Yes



شكل (A-11)

نقوم بكتابة البرنامج في نافذة المحرر كما بشكل (A-12)

Source Editor [SwitchTest *]

```
1  /*
2   * SwitchTest.java
3  */
4  public class SwitchTest extends javax.swing.JApplet {
5
6      /** Creates new SwitchTest */
7      public SwitchTest() {
8      }
9      int choice;    // user's choice of which shape to draw
10
11    // initialize applet by obtaining user's choice
12    public void init()
13    {
14       String input;  // user's input
15
16       // obtain user's choice
17       input = JOptionPane.showInputDialog(
18                "Enter 1 to draw lines\n" +
19                "Enter 2 to draw rectangles\n" +
20                "Enter 3 to draw ovals\n" );
21
```

3:2 INS
Test1 * | SwitchTest *

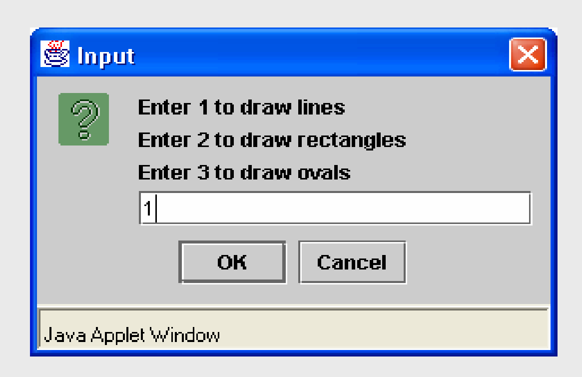
شكل(A-12)

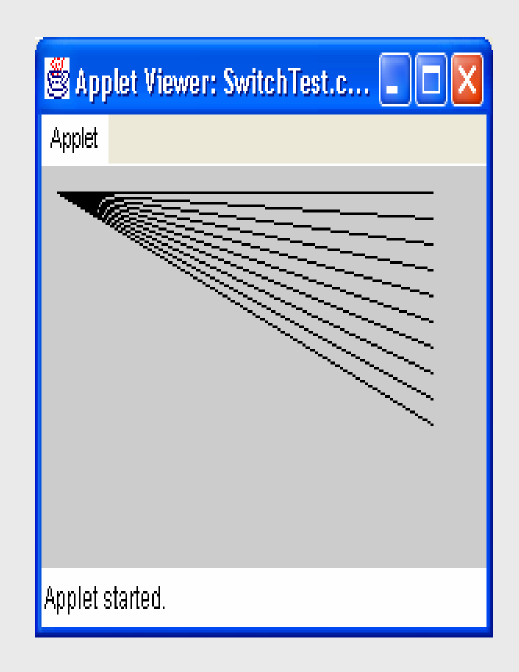
بعد كتابة البرنامج نقوم بعمل ترجمة له compile ثم بعد ذلك Build ثم بعد ذلك نقوم بالتنفيذ ونلاحظ أنه عند تنفيذ هذا البرنامج وهو أبلت يقوم برنامج الـ Forte بإنشاء ملف HTML ثم يقوم بتحميل الأبلت عليه ومن ثم تنفيذه من خلال عارض الأبلت appletviewer

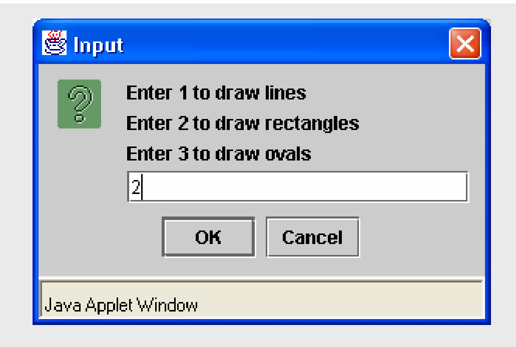
الشكل **(A-13)** يوضح البرنامج كاملا بالإضافة الى شكل التنفيذ:

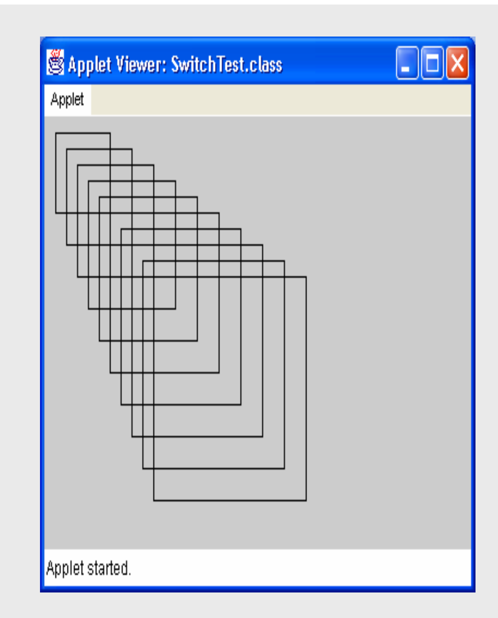
```
1.  // Fig. A.13: SwitchTest.java
2.  // Drawing lines, rectangles or ovals based on user input.
3.
4.  // Java core packages
5.  import java.awt.Graphics;
6.
7.  // Java extension packages
8.  import javax.swing.*;
9.
10. public class SwitchTest extends JApplet {
11. int choice;   // user's choice of which shape to draw
12.
13. // initialize applet by obtaining user's choice
14. public void init()
15. {
16. String input;  // user's input
17.
18. // obtain user's choice
19. input = JOptionPane.showInputDialog(
20. "Enter 1 to draw lines\n" +
21. "Enter 2 to draw rectangles\n" +
22. "Enter 3 to draw ovals\n" );
23.
24. // convert user's input to an int
25. choice = Integer.parseInt( input );
26. }
27.
28. // draw shapes on applet's background
29. public void paint( Graphics g )
30. {
31. // call inherited version of method paint
32. super.paint( g );
33.
34. // loop 10 times, counting from 0 through 9
35. for ( int i = 0; i < 10; i++ ) {
36.
37. // determine shape to draw based on user's choice
38. switch ( choice ) {
39.
40. case 1:
```

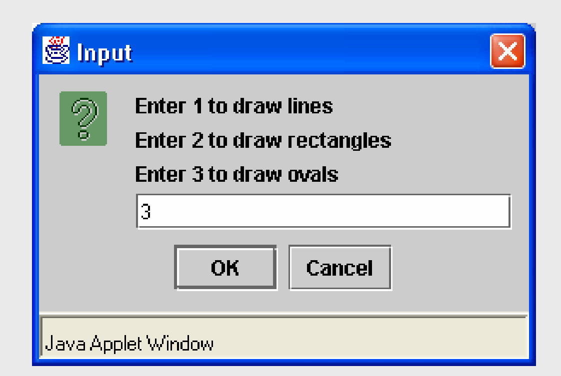
```
41. g.drawLine( 10, 10, 250, 10 + i * 10 );
42. break;  // done processing case
43.
44. case 2:
45. g.drawRect( 10 + i * 10, 10 + i * 10,
46. 50 + i * 10, 50 + i * 10 );
47. break;  // done processing case
48. case 3
49. g.drawOval( 10 + i * 10, 10 + i * 10,
50. 50 + i * 10, 50 + i * 10 );
51. break;  // done processing case
52. default:
53. g.drawString( "Invalid value entered",
54. 10, 20 + i * 15 );
55.
56. }  // end switch structure
57.
58. }  // end for structure
59.
60. }  // end paint method
61.
62. }  // end class SwitchTest
```

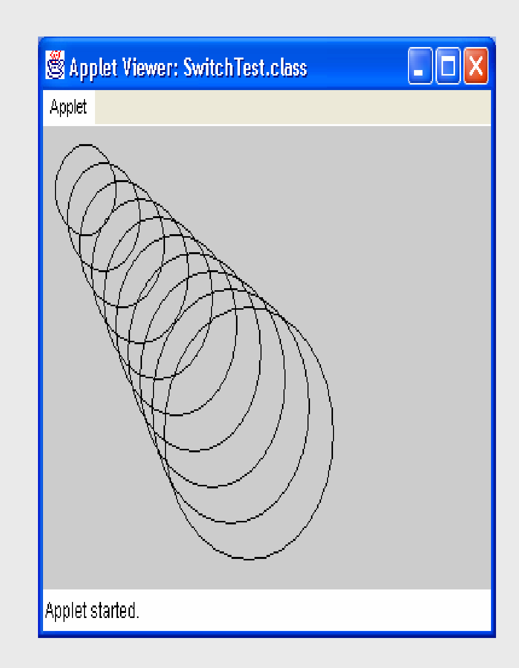












شكل (A-13) برنامج أبلت باستخدام برنامج الفورتي

ملحق ب

برامج الأبلت Applets

تعرضنا من خلال الوحدات السابقة إلى نوع من أنواع البرامج في لغة الجافا وهو التطبيقApplication وقلنا إنه يوجد نوع آخر من البرامج ألا وهو الأبلت Applet وتتميز هذه البرامج بإمكانية إدماجها داخل صفحات الويب، فمثلا عندما يتم تحميل صفحة ويب تحتوي على أبلت من خلال المتصفح فيقوم هذا المتصفح بتحميل الأبلت ويبدأ بتنفيذه.

متصفح الويب الذي يقوم بتنفيذ الأبلت يسمى حاوي الأبلت Applet container، تحتوي حزمة تطوير البرامج بالجافا Java 2 Software Development Kit على حاوي أبلت يسمى عارض الأبلت Applet Viewer وهو يستخدم لعمل اختبار للأبلت قبل دمجها مع صفحة الويب.

يوجد العديد من المتصفحات لا تدعم الجافا مباشرة مثل متصفح مايكروسوفت، يعتبر متصفح Netscape 6 أحد المتصفحات التي تدعم الجافا.

ملحوظة: لتنفيذ الأبلت على أحد المتصفحات التي لا تدعم الجافا نستخدم Java Plug-in (Converter) وسنتعرض له لاحقا.

مثال

```
1.  // Fig. B.1: WelcomeApplet.java
2.  // A first applet in Java.
3.
4.  // Java core packages
5.  import java.awt.Graphics;    // import class Graphics
6.
7.  // Java extension packages
8.  import javax.swing.JApplet;  // import class JApplet
9.
10. public class WelcomeApplet extends JApplet {
11.
12. // draw text on applet's background
13. public void paint( Graphics g )
14. {
15. // call inherited version of method paint
16. super.paint( g );
17.
18. // draw a String at x-coordinate 25 and y-coordinate 25
19. g.drawString( "Welcome to Java Programming!", 25, 25 );
20.
21. }  // end method paint
22.
23. }  // end class WelcomeApplet
```



شكل (B.1) برنامج أبلت وشكل التنفيذ

يوضح هذا البرنامج العديد من الخصائص الهامة للجافا ، لاحظ أن السطر رقم 19 هو الذي يقوم بالعمل الفعلي للبرنامج وهو رسم النص التالي على الشاشة

Welcome to Java Programming!

شرح البرنامج

السطور 1 – 2

```
// Fig. B.1: WelcomeApplet.java
// A first applet in Java.
```

كما قلنا سابقا أي سطر يبدأ بـ // يعتبر ملاحظة أي لا يدخل ضمن البرنامج ولكن يستخدم للتوضيح للمبرمج وهنا السطر الأول يوضح اسم البرنامج ورقم الشكل كما أن السطر الثاني يوضح الهدف من البرنامج.

السطر رقم 5

```
import java.awt.Graphics;    // import class Graphics
```

قلنا سابقا إن لغة الجافا تحتوي على مكونات معرفة سابقا تسمى فصول classes وهذه الفصول مجمعة داخل حزم packages . والسطر رقم 5 هو عبارة عن جملة import التي تقول للمترجم أن يحمل الفصل Graphics من الحزمة java.awt . الفصل Graphics يسمح للأبلت أن تقوم برسم أشكال مثل خط ، مستطيل ، شكل دائري ، سلسلة من الحروف ، ...الخ.

السطر رقم 8

```
Import javax.swing.JApplet; // import class JApplet
```

هو أيضا عبارة عن جملة import والتي تخبر المترجم أن يقوم بتحميل الفصل JApplet من الحزمة javax.swing . نقوم بدمج هذا الفصل عادة عندما تقوم بانشاء أبلت.

ملحوظة: يوجد إصدار قديم من هذا الفصل يسمى Appletوموجود في الحزمة java.applet

كما هو الحال في برامج التطبيقات فإن كل أبلت تحتوي على الأقل على تعريف لفصل واحد وهذا الفصل لابد وأن يكون امتداد لفصل آخر موجود من قبل بمعنى أن الفصل لا ينشأ من الصفر ولكن ينشأ كتكملة وامتداد لفصل آخر وذلك نراه في تعريف الفصل في الكلمة extends ثم يتبعها اسم الفصل الأساسي ولكن إذا لم نكتب هذه الكلمة ثم اسم الفصل الأساسي وذلك في برامج التطبيقات اعتبر المترجم ضمنيا أن الفصل امتداد للفصل Object أما في الأبلت فيجب كتابة هذه الكلمة ويتبعها اسم الفصل JavaApplet كما في السطر رقم 10

```
Public class WelcomeApplet extends JApplet {
```

وهو تعريف الفصل WelcomeApplet . في نهاية هذا السطر يوجد القوس الأيسر } والقوس الأيمن له موجود في السطر رقم 23 وبينهما توجد تعليمات الفصل

في هذه العلاقة الوراثية يسمى الفصل JApplet بالفصل السوبر أو الأساسيSuperclass كما يسمى الفصل WelcomeApplet بالفصل الفرعي subclass ، يرث الفصل الفرعي كل خصائص الفصل السوبر كما يرث أيضا جميع الطرق التي به بالإضافة الخصائص والطرق الخاصة بالفصل الفرعي نفسه وهي على سبيل المثال قدرة الفصل WelcomApplet على رسم النص Welcome To Java Programming ! على الأبلت.

سؤال : لماذا دائما الفصل أبلت يكون امتداد لفصل آخر وهو JApplet ؟

الإجابة: لكي يقوم عارض الأبلت أو المتصفح بعرض الأبلت يحتاج على الأقل إلى 200 طريقة ونلاحظ في المثال السابق أنه يحتوي على طريقة واحدة فقط وذلك لأنه ورث الطرق الأخرى من الفصل السوبر، فإذا كنا في كل مرة نكتب أبلت نحتاج لعمل 200 طريقة فإننا لن نعمل أبلت أبدا.

السطر رقم 13

Public void paint(Graphics g)

هـذا السـطر يبـدأ بتعريـف الطريقـة paint وهـي واحـدة مـن ثـلاث طـرق أخـرى يقـوم حـاوي الأبلـت باستدعائهم عند تنفيذ الأبلت وهم (init , start , paint) وهذه الطرق الثلاث تورث من قبل الفصل السوبر الى الفصل الفرعي ، إذا لم تقم بتعريف أحد هذه الطرق مرة أخرى في الأبلت يقوم حاوي الأبلت باستدعاء النسخة الموروثة .

ملحوظة: النسخة الموروثة من الطريقة init والطريقة start لا تحتوي على تعليمات لذلك فهي لا تقوم بأي مهمة كما أن النسخة الموروثة من الطريقة paint لا تقوم بعرض أي رسوم على الأبلت .

لكي نجعل الأبلت تقوم بعرض رسوم فإننا نقوم بإعادة تعريف الطريقة paint وإضافة لها جملة الرسم

السطور 13 - 14

تحتوي على تعريف الطريقة paint وتعليماتها ، وكما هو الحال عند عرض صندوق رسالة فإننا كنا نقوم باستدعاء الطريقة showMessageDialog والموجودة في الفصل JoptionPane فإننا هنا لكي نقوم بعرض رسوم على الأبلت نستدعي الطريقة paint ولكن المبرمج لا يقوم باستدعائها صراحة ولكن حاوي الأبلت هو الذي يقوم باستدعائها لكي تجعل الأبلت يعرض رسوماً ويقوم حاوي الأبلت أيضا بتمرير المعلومات التي تحتاجها في الرسم وهي الهدف Graphics ويسمى بـ g ، تستخدم الطريقة paint الهدف Graphics لكي تقوم برسم الأشكال والرسوم على الأبلت .

لاحظ أن الطريقة معرفة على أنها public لكي يستطيع حاوي البلت استدعاء الطريقة paint لذلك يجب أن تكون كل الطرق public

السطر رقم 16

```
super.paint( g );
```

هذا السطر يقوم باستدعاء النسخة الأصلية الموجودة في الفصل السوبر JApplet

السطر رقم 19

```
g.drawString( "Welcome to Java Programming!", 25, 25 );
```

كما قلنا سابقا إن هذا السطر هو الذي يقوم فعليا برسم النص على الأبلت فهو يستدعي الطريقة drawstring والموجودة داخل الهدف Graphics المسمى بـ g لذلك فإننا نستدعيها بأن يكتب اسم الهدف يتبعها اسم الطريقة وتفصلهما نقطة.

أول عنصر داخل الطريقةdrawString هو النص نفسه وهو
Welcome To Java Progamming !
ثاني عنصر هو إحداثي المحور السيني و إحداثي المحور الصادي الذي سوف نبدأ منهما الرسم على الأبلت وهما في المثال 25, 25 مع ملاحظة أن الاحداثي 0 , 0 يبدأ عند الركن العلوي في اليسار.

بعد عملية الترجمة compilation وقبل أن نستطيع تنفيذ الأبلت لابد أولا من إنشاء ملف HTML لكي يقوم بتحميل الأبلت إلى حاوي الأبلت وهو إما أن يكون المتصفح أو عارض الأبلت appletviewer وملف الـ HTML يكون له امتداد html. أو htm. ولكي نقوم بالتنفيذ لابد أن يشير ملف الـ HTML الى اسم الأبلت . والمثال (B-2) يوضح كيفية تعريف اسم الأبلت داخل ملف الـ HTML .

```
1.	<html>
2.	<applet code = "WelcomeApplet.class" width = "300" height = "45">
3.	</applet>
4.	</html>
```

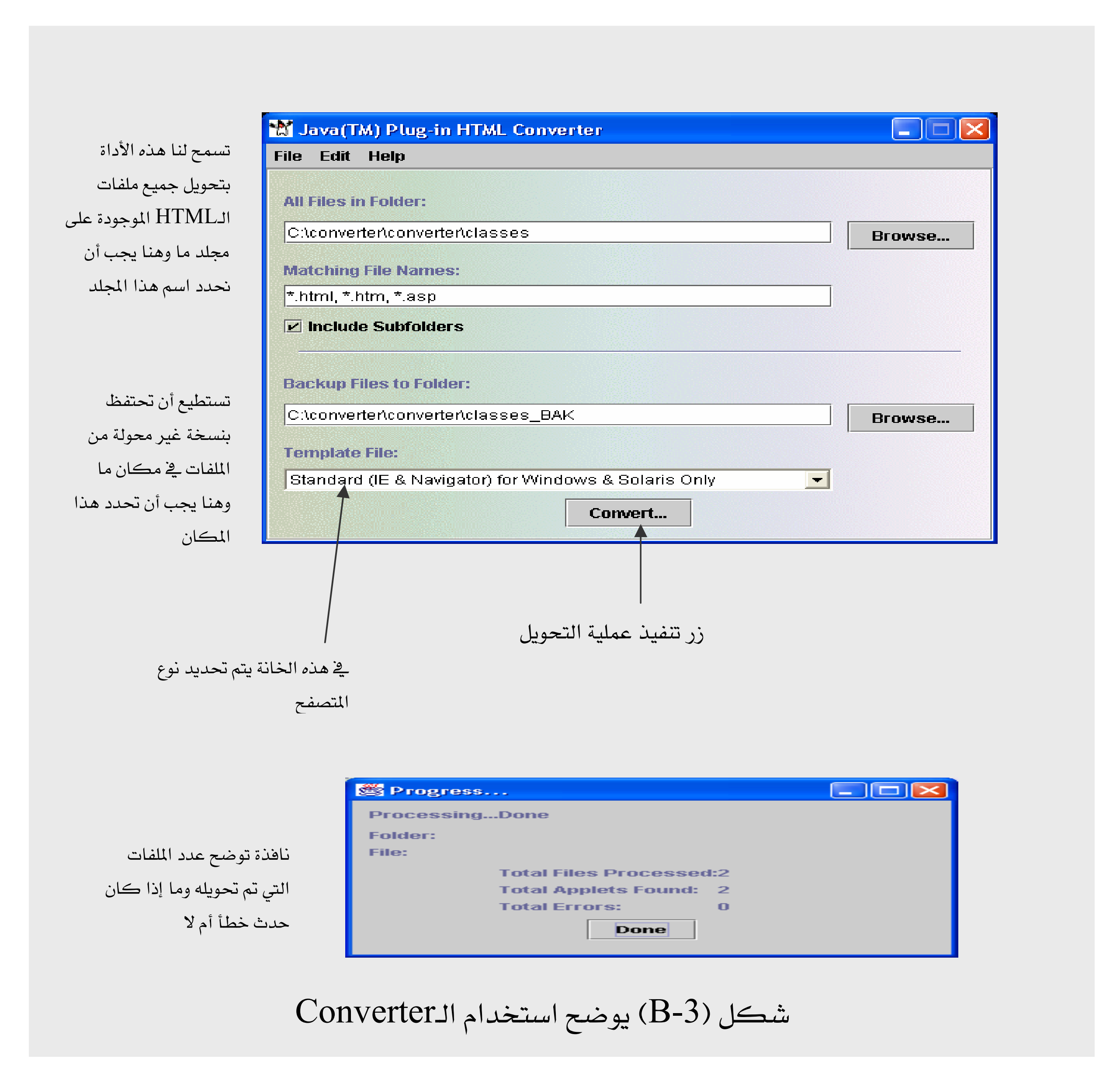
شكل (B-2) ملفHTML وبه اسم الأبلت

لاحظ أن السطر رقم 2 معرف فيه اسم الفصل الأبلت وهو WelcomeApplet.class كما هو محدد في عرض وارتفاع الأبلت التي ستظهر في المتصفح (حاوي الأبلت)

ملحوظة: معظم برامج التحرير للجافا تقوم هي بإنشاء ملف الـ HTML نيابة عن المبرمج وذلك عند التنفيذ.

عرض الأبلت على المتصفحات التي لا تدعم الجافا

لعرض أي أبلت على متصفح لا يدعم الجافا نستخدم ما يسمى بـ
Java plug-in HTML converter
وهي أداة تم تطويرها من قبل شركة صن وتستخدم لتحويل ملف HTML المحمل عليه الأبلت إلى ملف آخر بنفس الاسم الامتداد يمكن أن يعرض من خلال المتصفحات التي لا تدعم الجافا.
هذه الأداة موجودة مجانا على موقع الشركة على الإنترنت، بمجرد عمل تحميل لها وتركيبها على الكمبيوتر يمكن أن نشغلها من خلال الملف HTMLConverter.bat ويتم تنفيذ هذا الملف من على محث الدوس طريقة التشغيل. انظر الشكل (B-3):



شكل (B-3) يوضح استخدام الـConverter

References

أولا : المراجع العربية :

١ - **سويلم محمد نبهان**
تحليل وتصميم نظم المعلومات
المكتبة الأكاديمية – القاهرة - ١٩٩٦م

٢ - **علي على يوسف**
تحليل وتصميم نظم المعلومات
خوارزم – القاهرة – فبراير ١٩٩٨

٣ - **د. عوض منصور & د. محمود نحاس**
برمجة با سكال وتيربو باسكال لطلبة الهندسة والعلوم
شبكة الكمبيوتلر الشخصي - مؤسسة الجاسم للإلكترونيات، ١٩٨٧م

ثانيا : المراجع الأجنبية :

١ - Wilson, Thomas C and Shortt Joseph, “ Pascal from begin to end”,

٢ - Deitel and Deitel , “ Java How to Program”, Prentice Hall, 2002

٣ - Liang Y. Daniel, " Introduction to Java Programming", Que E&T, 1999

الموضوع	الصفحة

تقدر المؤسسة العامة للتعليم الفني والتدريب المهني الدعم

المالي المقدم من شركة بي آيه إي سيستمز (العمليات) المحدودة

GOTEVOT appreciates the financial support provided by BAE SYSTEMS

BAE SYSTEMS



المملكـة العربيـة السـعودية
المؤسسة العامة للتعليم الفني والتدريب المهني
الإدارة العامة لتصميم وتطوير المناهج

البرمجيات

برمجة ٢

١٤٢ حاب

مقدمه

الحمد لله وحده، والصلاة والسلام على من لا نبي بعده، محمد وعلى آله وصحبه، وبعد:

تسعى المؤسسة العامة للتعليم الفني والتدريب المهني لتأهيل الكوادر الوطنية المدربة القادرة على شغل الوظائف التقنية والفنية والمهنية المتوفرة في سوق العمل، ويأتي هذا الاهتمام نتيجة للتوجهات السديدة من لدن قادة هذا الوطن التي تصب في مجملها نحو إيجاد وطن متكامل يعتمد ذاتياً على موارده وعلى قوة شبابه المسلح بالعلم والإيمان من أجل الاستمرار قدماً في دفع عجلة التقدم التنموي؛ لتصل بعون الله تعالى لمصاف الدول المتقدمة صناعياً.

وقد خطت الإدارة العامة لتصميم وتطوير المناهج خطوة إيجابية تتفق مع التجارب الدولية المتقدمة في بناء البرامج التدريبية، وفق أساليب علمية حديثة تحاكي متطلبات سوق العمل بكافة تخصصاته لتلبي متطلباته ، وقد تمثلت هذه الخطوة في مشروع إعداد المعايير المهنية الوطنية الذي يمثل الركيزة الأساسية في بناء البرامج التدريبية، إذ تعتمد المعايير في بنائها على تشكيل لجان تخصصية تمثل سوق العمل والمؤسسة العامة للتعليم الفني والتدريب المهني بحيث تتوافق الرؤية العلمية مع الواقع العملي الذي تفرضه متطلبات سوق العمل، لتخرج هذه اللجان في النهاية بنظرة متكاملة لبرنامج تدريبي أكثر التصاقاً بسوق العمل، وأكثر واقعية في تحقيق متطلباته الأساسية.

وتتناول هذه الحقيبة التدريبية " برمجة ٢ " لمتدربي قسم" البرمجيات " للكليات التقنية موضوعات حيوية تتناول كيفية اكتساب المهارات اللازمة لهذا التخصص.

والإدارة العامة لتصميم وتطوير المناهج وهي تضع بين يديك هذه الحقيبة التدريبية تأمل من الله عز وجل أن تسهم بشكل مباشر في تأصيل المهارات الضرورية اللازمة، بأسلوب مبسط يخلو من التعقيد، وبالاستعانة بالتطبيقات والأشكال التي تدعم عملية اكتساب هذه المهارات.

والله نسأل أن يوفق القائمين على إعدادها والمستفيدين منها لما يحبه ويرضاه؛ إنه سميع مجيب الدعاء.

الإدارة العامة لتصميم وتطوير المناهج



برمجة ٢

المصفوفـــات

المصفوفـــات

١

الجـــــدارة:

معرفة كيفية استخدام المصفوفات لحل بعض المشاكل البرمجية.

الأهـــــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١ - معرفة الغاية من استخدام المصفوفات.
٢ - تعريف المصفوفات وحجز المواقع لها.
٣ - إعطاء المصفوفات القيم الابتدائية عند التعريف.
٤ - الوصول لموقع معين داخل المصفوفة لتعديل محتوياته.
٥ - ترتيب عناصر المصفوفات.
٦ - معرفة طرق البحث عن عنصر معين داخل المصفوفات.
٧ - التعامل مع المصفوفات ذات البعدين.
٨ - كتابة البرامج التي تستخدم المصفوفات لحل المشاكل البرمجية.

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ١٠ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

مقدمـة :

في هذه الوحدة سوف يتم التطرق للمصفوفات ذات البعد الواحد والمصفوفات ذات البعدين، حيث سنقوم بشرح كيفية تعريف المصفوفات وحجز المواقع لها مع توضيح كيفية إعطاء القيم الابتدائية للمصفوفات عند تعريفها. كما وسنقوم بشرح عمليات ترتيب عناصر المصفوفات والبحث عن عنصر معين أو عدة عناصر في المصفوفات. وفي نهاية هذا الفصل هنالك عدد من التمارين المتعلقة بالمصفوفات.

تعريف المصفوفات وحجز المواقع لها :

المصفوفات هي عبارة عن مواقع يتم تخزين البيانات فيها لمدة مؤقتة (طيلة فترة تنفيذ البرنامج فقط)، وعند تعريف المصفوفة وإنشاءها يتم حجز عدد محدد من المواقع المتجاورة في الذاكرة لتخزين البيانات فيها، حيث يتم الوصول للبيانات المخزنة في هذه المواقع عن طريق اسم المصفوفة ورقم الموقع (Index). والغاية من استخدام المصفوفات هي تخزين عدد غير محدد من القيم تحت اسم واحد فقط (اسم المصفوفة) دون الحاجة إلى تخزين كل قيمة في متغير (Variable) منفصل.

لاستخدام المصفوفات في البرنامج لابد من تعريفها وحجز المواقع لها. حيث يتم ذلك كما يلي:

```
1. int array1[];
2. array1[] = new int[9];
```

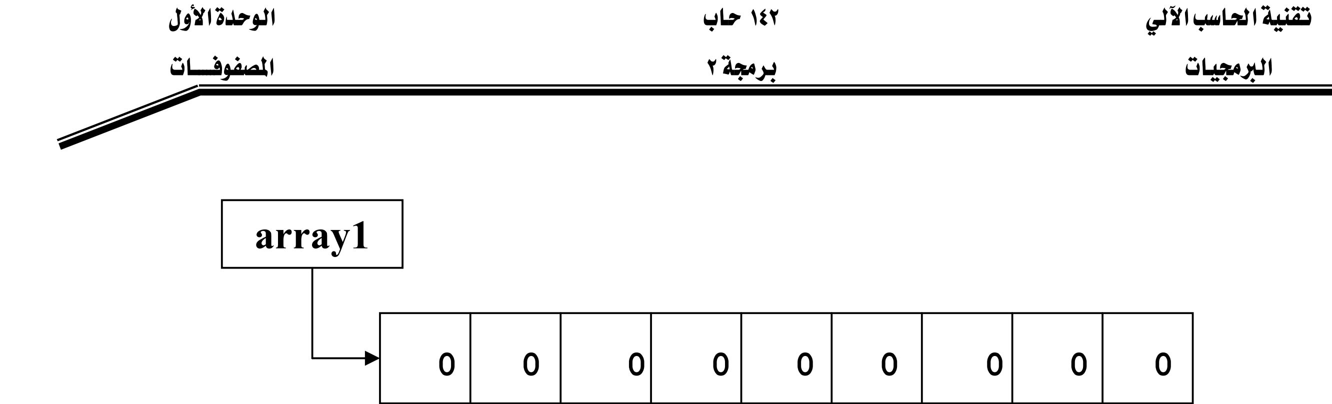
في السطر رقم (١) تم تعريف المصفوفة array1 من نوع int ، أي أننا نستطيع تخزين أعداد من نوع int في هذه المصفوفة. بينما في السطر رقم (٢) تم حجز ٩ مواقع لهذه المصفوفة (من الموقع رقم صفر إلى الموقع رقم ٨) لنستطيع تخزين ٩ أعداد صحيحة على الأكثر في هذه المصفوفة. كما ويمكن دمج الجملتين السابقتين بجملة واحدة لتصبح كما يلي:

```
int array1[] = new int[9];
```

ويمكن كتابة الجملة السابقة بالشكل التالي:

```
int[] array1 = new int[9];
```

يتم حجز المواقع للمصفوفة array1 كما في الشكل (١-١):



شكل (١-١)

في لغة جافا، رقم موقع العنصر في المصفوفة يكتب بين أقواس مربعة بعد اسم المصفوفة (مثال: array1[k]، حيث أن k يمثل رقم الموقع في المصفوفة، وفي مثالنا السابق هو عدد صحيح محصور بين الصفر والثمانية). وبشكل عام، عند حجز n من المواقع للمصفوفة فإن أرقام هذه المواقع تكون من صفر ولغاية n-1.

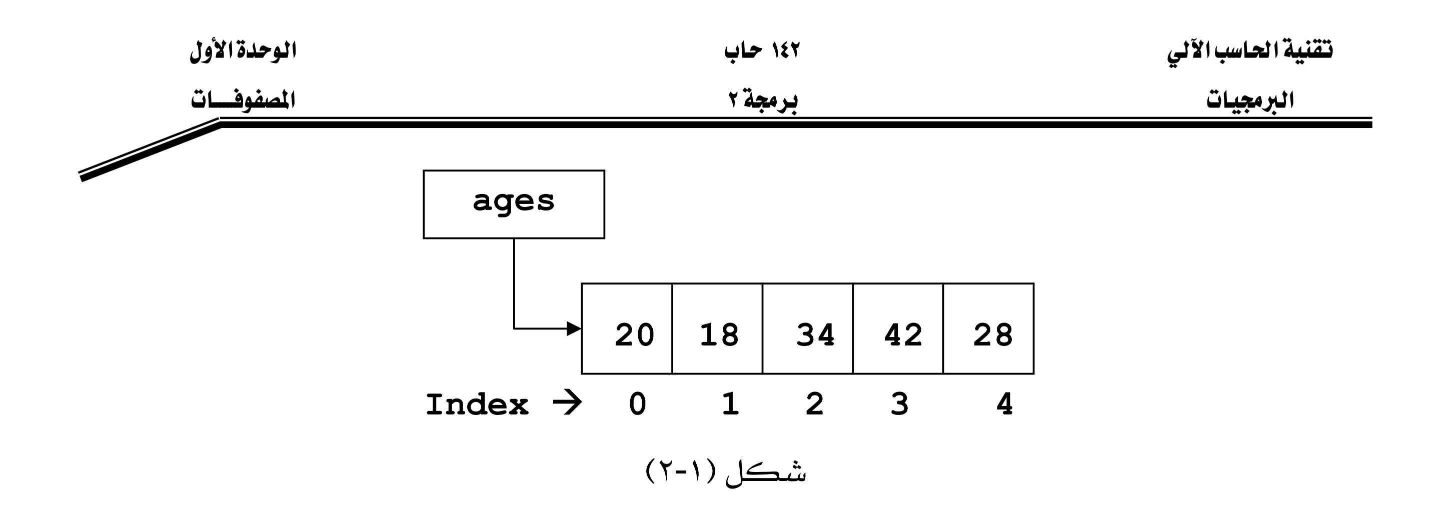
ونستطيع بشكل اختياري أن نحدد للمصفوفة قيماً ابتدائية يتم تحديدها عند تعريف المصفوفة، وإذا لم نحدد للمصفوفة قيماً ابتدائية فإنه يتم تخزين القيمة التلقائية (Default Value) لنوع المصفوفة وذلك عند حجز المواقع لها. والقيم التلقائية للأنواع هي كما يلي:

int, byte, short, long	→	0
double, float	→	0.0
char	→	\u0000 فراغ
String	→	null
Boolean	→	false

ويمكن تحديد القيم الابتدائية للمصفوفة بالطريقة التالية:

```
int ages[]= {20, 18, 34, 42, 28};
```

من خلال هذه الجملة قمنا بتعريف مصفوفة اسمها ages، وخزّنا فيها قيماً ابتدائية، حيث سيتم حجز مواقع على عدد هذه القيم الابتدائية. والشكل (١-٢) يوضح عملية التخزين.



شكل (١-٢)

وللوصول للرقم 42 في المصفوفة ages يجب استخدام الشكل التالي: ages[3]، حيث نستطيع طباعة الرقم 42 عن طريق الجملة التالية:

```
System.out.println(ages[3]);
```

ولتعديل القيمة المخزّنة في الموقع رقم 1 لتصبح 53 عوضاً عن 18، يجب تنفيذ الجملة التالية:

```
ages[1]=53;
```

مثال: ١-١:

```
// array1.java
1.    public class array1{
2.     public static void main(String args[]){
3.      double a[]=new double[10];
4.      for(int i=0; i<10; i++){
5.       a[i]=i*2.5;
6.       System.out.println(a[i]);
7.      } // end for
8.     } // end main
9.    } // end class array1
```

شرح المثال:

في السطر رقم (٣) قمنا بتعريف مصفوفة اسمها a من نوع double وحجزنا لها ١٠ مواقع. وفي السطر رقم (٥) تم تخزين ناتج العملية التالية في مواقع المصفوفة i*2.5 حيث تتغير قيمة i من صفر ولغاية تسعة لتحدد رقم الموقع المراد تخزين ناتج العملية فيه ولتؤثر على ناتج العملية. وفي السطر رقم (٦) قمنا بطباعة محتويات المصفوفة a. والشكل (١-٣) يبين نتائج البرنامج السابق:

```
C:\Program Files\Xinox Software\JCreatorV3 LE\GE2001.exe
 0.0
 2.5
 5.0
 7.5
 10.0
 12.5
 15.0
 17.5
 20.0
 22.5
Press any key to continue...
```

شكل (١-٣)

مثال: ١-٢:

```
// array2.java
1.   import javax.swing.*;
2.    public class array2{
3.     public static void main(String args[]){
4.     int b[]=new int[5];
5.     String s;
6.     for(int i=0; i<5; i++){
7.      s=JOptionPane.showInputDialog("Enter a number:");
8.      b[i]=Integer.parseInt(s);
9.     } // end for
10.    for(int i=0; i<5; i++)
```

```
11.    if(b[i]>=60)
12.      System.out.println(b[i]);
13.  } // end main
14.  } // end class array2
```

شرح المثال:

في السطر رقم (٤) قمنا بتعريف مصفوفة اسمها b وتم حجز ٥ مواقع لهذه المصفوفة. في الأسطر (٦-٩) يتم إدخال قيم ليتم تخزينها في المصفوفة b. وفي الأسطر (١٠-١٢) تتم عملية طباعة محتويات المصفوفة b. حيث يقوم هذه البرنامج بطلب المستخدم إدخال ٥ درجات لتخزينها في المصفوفة b، بعد ذلك يقوم البرنامج بطباعة الدرجات التي تزيد عن أو تساوي ٦٠. والشكل (١-٤) تبين عمليات إدخال الدرجات:

Input
Enter a number:
78
OK Cancel

Input
Enter a number:
47
OK Cancel

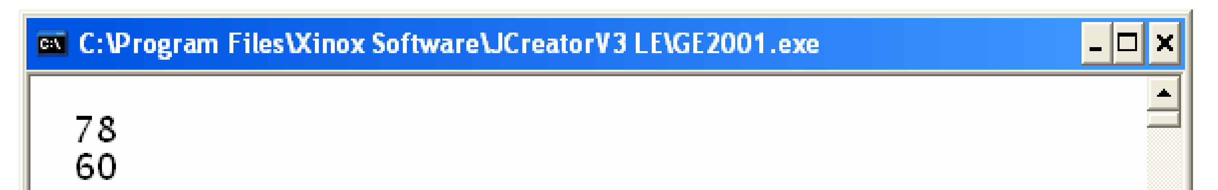
Input
Enter a number:
60
OK Cancel

Input
Enter a number:
93
OK Cancel

Input
Enter a number:
72
OK Cancel

شكل (١-٤)

بينما يبين الشكل (١-٥) نتائج البرنامج السابق:



شكل (١-٥)

ملاحظـات مهمـة:

١ - يجب أن يكون رقم الموقع (Index) عند التعامل مع المصفوفة عدداً صحيحاً موجباً.

٢ - يجب أن لا نتجاوز عدد المواقع المحجوزة للمصفوفة عند استخدامها.

٣ - إذا لم نحدد قيماً ابتدائية للمصفوفة فيجب أن نستخدم الكلمة المحجوزة (new) لحجز مواقع للمصفوفة كما ذكر سابقاً.

٤ - إذا لم تعطى المصفوفة قيماً ابتدائية عند تعريفها فإنها تأخذ القيم التلقائية (Default Value) كقيم ابتدائية وذلك حسب النوع (Type) الذي حدد للمصفوفة. كما ذكر سابقاً.

٥ - نستطيع معرفة عدد المواقع المحجوزة للمصفوفة من خلال كتابة اسم المصفوفة ثم نقطة ثم length. (مثال: array1.length ، من خلال هذه الجملة نستطيع معرفة عدد المواقع المحجوزة للمصفوفة array1).

٦ - نستطيع استخدام أحد الأشكال التالية لتعريف وحجز مواقع المصفوفة:

```
1. int array[]=new int[5];
```

```
2. int array[];
  array=new int[5];
```

```
3. int [] array = new int[5];
```

```
4. int [] array;
   array=new int[5];
```

٧ - نستطيع استخدام أحد الأشكال التالية لتخزين القيم الابتدائية في المصفوفة:

```
1. int array[]={5, 3, 8, 9, 2};
```

```
2. int array[]=new int[] {5, 3, 8, 9, 2};
```

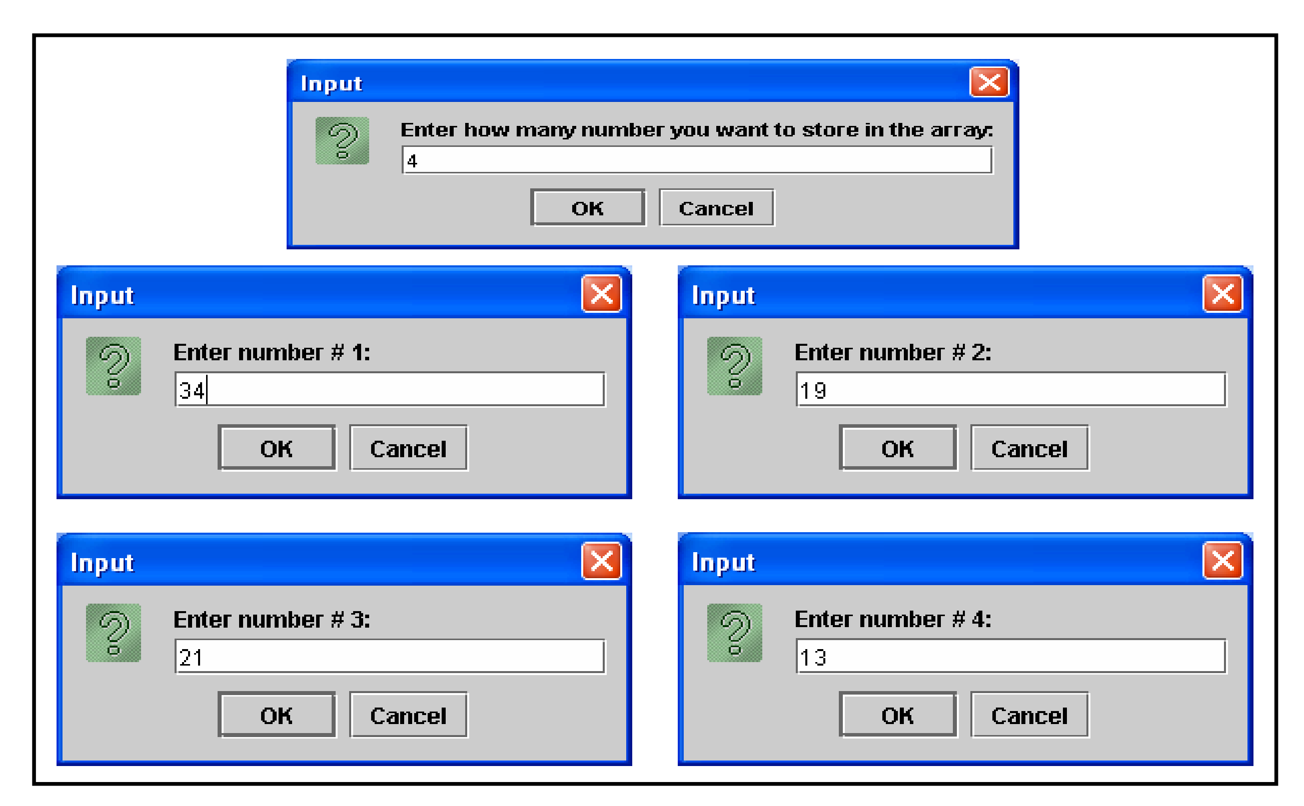
أمثلة على استخدام المصفوفات:

مثال: ١-٣:

```
//array3.java
1.   import javax.swing.*;
2.   class array3{
3.     public static void main(String args[]){
4.      String s, output, title, str1, str2;
5.      str1="Enter how many number you want to store in the array:";
6.      str2="Enter number # ";
7.      int n, odd=0;
8.      s=JOptionPane.showInputDialog(str1);
9.      n=Integer.parseInt(s);
10.     int [] arr=new int[n];
11.     output= " ";
12.     for(int i=0; i<arr.length; i++){
13.      s=JOptionPane.showInputDialog(str2+(i+1)+":");
14.      arr[i]=Integer.parseInt(s);
15.      output+=arr[i]+"\n ";
16.     }  //end for
17.     for(int i=0; i<arr.length; i++)
18.     if(arr[i]%2==1) odd++; // end for
19.     title="The results of the example (1-3)";
20.     output+="\nThere are "+odd+" odd numbers in the array";
21.     JOptionPane.showMessageDialog(null, output, title,
            JOptionPane.INFORMATION_MESSAGE);
22.     System.exit(0);
23.   }       //end main
24.  } //end class array3
```

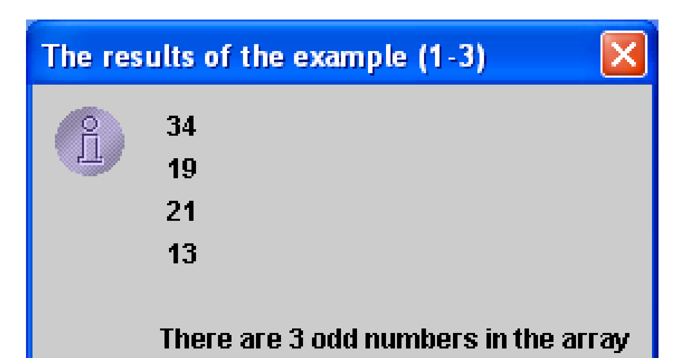
شرح المثال:

في الأسطر (٨-٩) يطلب البرنامج إدخال رقم، حيث يحوّل هذا العدد إلى رقم صحيح ويخزّن في المتغير n. في السطر رقم (١٠) يتم تعريف المصفوفة arr وحجز عدد n من المواقع لهذه المصفوفة (أي يتم تحديد عدد العناصر المحجوزة للمصفوفة عن طريق المستخدم للبرنامج). في الأسطر (١٢-١٦) يتم إدخال n من الأرقام وتخزّن في المصفوفة arr (لاحظ الدوران يبدأ من صفر ولغاية أقل من arr.length). في السطر رقم (١٨) يتم فحص الأرقام المخزنة في المصفوفة arr، حيث إذا كان الرقم فرديا يضاف واحد للمتغير odd (متغير يخزّن فيه عدد الأرقام الفردية) . وفي السطر رقم (٢١) يتم طباعة نتائج البرنامج. الشكل (١-٦) يبين عمليات الإدخال في البرنامج:



شكل (١-٦)

بينما يعرض الشكل (١-٧) مخرجات البرنامج السابق:



شكل (١-٧)

مثال: ١-٤:

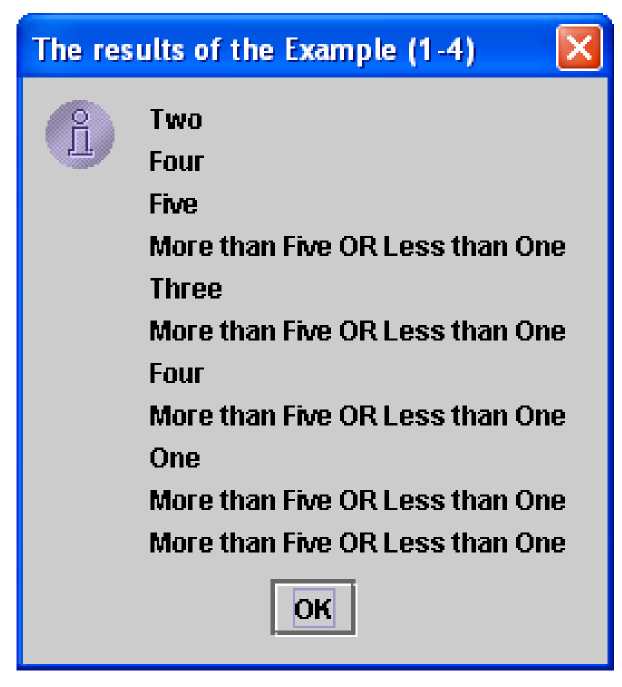
```
// array4.java

1.   import javax.swing.*;
2.   class array4{
3.    public static void main(String args[]){
4.     int a[]={2, 4, 5, -5, 3, 10, 4, 11, 1, 7, -2};
5.     String title="The results of the Example (1-4)";
6.     String results="";

7.     for(int i=0; i<a.length; i++)
8.     {
9.      switch(a[i]){
10.     case 1:  results=results+"One\n"; break;
11.     case 2:  results=results+"Two\n"; break;
12.     case 3:  results=results+"Three\n"; break;
13.     case 4:  results=results+"Four\n"; break;
14.     case 5:  results=results+"Five\n"; break;
15.     default: results=results+"More than Five OR Less than One\n";
16.     }
17.    }
18.    JOptionPane.showMessageDialog(null, results, title,
           JOptionPane.INFORMATION_MESSAGE);
19.    System.exit(0);
20.   }
21.  }
```

شرح المثال:

في السطر رقم (٤) تم تعريف المصفوفة a وإعطائها قيماً ابتدائية. في الأسطر (٧-١٧) دوران للتعامل مع جميع عناصر المصفوفة من خلال جملة switch، حيث سيتم طباعة الرقم بالأحرف إذا كان هذا الرقم بين واحد وخمسة وتطبع جملة "More than Five OR Less than One" إذا كان الرقم غير ذلك. والشكل (١-٨) يبين مخرجات البرنامج السابق:



شكل (١-٨)

مثال: ١-٥:

```
// array5.java

1.     import javax.swing.*;
2.     class array5{
3.      public static void main(String args[]){
4.       double marks[] = new double[6];
5.       String names[] = new String[6];
6.       String s;
7.       String t1 = "Enter the Student's name:";
8.       String t2 = "Enter his mark:";
```

```
9.      for(int i=0; i<6; i++){
10.      s=JOptionPane.showInputDialog(t1);
11.      names[i]=s;
12.      s=JOptionPane.showInputDialog(t2);
13.      marks[i]=Double.parseDouble(s);
14.     }
15.     String title = "The passed students";
16.   String results="The following students are passed the exam:\n";
17.     for(int i=0; i<6; i++){
18.      if(marks[i]>=60)
19.        results=results+names[i]+"\n";
20.     }
21.   JOptionPane.showMessageDialog(null, results,  title,
          JOptionPane.INFORMATION_MESSAGE);
22.     System.exit(0);
23.    }
24.   }
```

شرح المثال:

في السطر رقم (٤) تم تعريف مصفوفة اسمها marks من نوع double وحجز ٦ مواقع لها، حيث ستخزن الدرجات في هذه المصفوفة. بينما في السطر رقم (٥) تم تعريف مصفوفة اسمها names من نوع String وحجز ٦ مواقع لها حيث ستستخدم هذه المصفوفة لتخزين أسماء الطلاب. في الأسطر (٩-١٤) استخدم الدوران لإدخال أسماء ودرجات الطلاب الستة وتخزينها في المصفوفات الخاصة بها. في الأسطر (١٧-٢٠) ومن خلال جملة الدوران يتم إضافة أسماء الطلاب الذين تزيد درجاتهم عن أو تساوي ٦٠ إلى المخرجات. وفي السطر (٢١) يتم طباعة المخرجات والتي تحتوي على أسماء الطلاب الذين تزيد درجاتهم عن أو تساوي ٦٠. والشكل (١-٩) يبين عمليات الإدخال في البرنامج:


Input
Enter the Student's name:
Ahmad
OK
Cancel
Input
Enter his mark:
72
OK
Cancel
Input
Enter the Student's name:
Saoud
OK
Cancel
Input
Enter his mark:
47
OK
Cancel
Input
Enter the Student's name:
Khaled
OK
Cancel
Input
Enter his mark:
57
OK
Cancel
Input
Enter the Student's name:
Saleh
OK
Cancel
Input
Enter his mark:
45
OK
Cancel

شكل (١-٩)

Input
Enter the Student's name:
Ali
OK Cancel

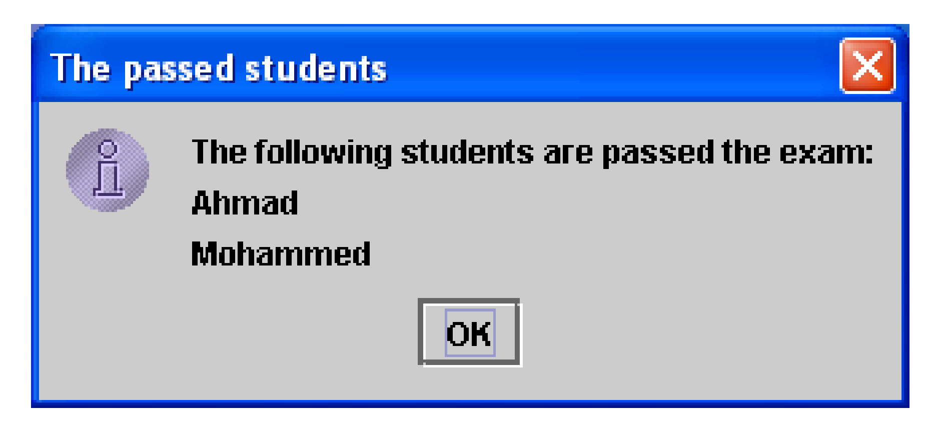
Input
Enter his mark:
53
OK Cancel

Input
Enter the Student's name:
Mohammed
OK Cancel

Input
Enter his mark:
92
OK Cancel

تكملة الشكل (١-٩)

والشكل (١-١٠) يبين مخرجات البرنامج السابق:



شكل (١-١٠)

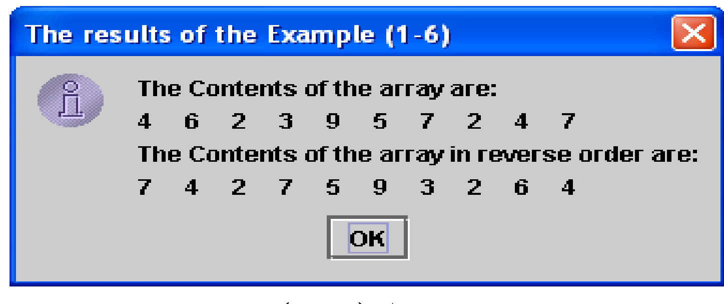
مثال: ١-٦:

```
// array6.java

1.     import javax.swing.*;
2.     class array6{
3.      public static void main(String args[]){
4.       int num[] = new int[]{4, 6, 2, 3, 9, 5, 7, 2, 4, 7};
5.       String output="The Contents of the array are:\n";
6.       String title="The results of the Example (1-6)";
7.       for(int i=0; i<=num.length-1; i++)
8.        output+=num[i]+"     ";
9.       output+="\nThe Contents of the array in reverse order are:\n";
10.      for(int i=num.length-1; i>=0; i--)
11.       output+=num[i]+"     ";
12.  JOptionPane.showMessageDialog(null, output,  title,
          JOptionPane.INFORMATION_MESSAGE);
13.      System.exit(0);
14.     }
15.    }
```

شرح المثال:

في الأسطر (١٠-١١) تم إحداث دوران عكسي وذلك لإضافة عناصر المصفوفة إلى المخرجات وبشكل عكسي. حيث يقوم هذا البرنامج بطباعة محتويات المصفوفة num بشكل عادي وبشكل عكسي (بدءاً بالموقع الأخير في المصفوفة num.length-1 وانتهاءً بالموقع رقم صفر). والشكل (١-١١) يبين مخرجات البرنامج السابق:



شكل (١-١١)

ترتيب عناصر المصفوفة (Sorting):

في كثير من التطبيقات والبرامج قد نحتاج إلى ترتيب محتويات المصفوفات. وترتيب المصفوفات أما أن يكون ترتيباً تصاعدياً من الأصغر إلى الأكبر أو يكون تنازلياً من الأكبر إلى الأصغر. وهنالك عدد من خوارزميات الترتيب المستخدمة في ترتيب العناصر وسوف نستخدم الترتيب الفقاعي (Bubble Sort).

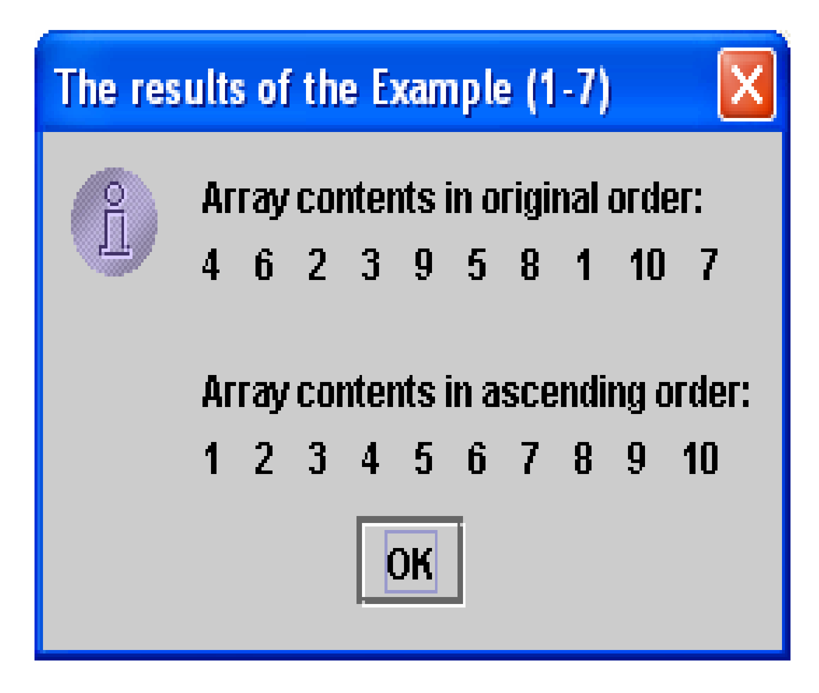
مثال ١-٧:

```
// array7.java

1.     import javax.swing.*;
2.     class array7{
3.      public static void main(String args[]){
4.       int num[] = new int[] {4, 6, 2, 3, 9, 5, 8, 1, 10, 7};
5.       int temp;
6.       String title="The results of the Example (1-7)";
7.       String output="Array contents in original order:\n";
8.       for(int i=0; i<num.length; i++)
9.        output+=num[i]+"    ";
10.      for(int i=1; i<num.length; i++)
11.       for(int j=0; j<num.length-1; j++)
12.        if(num[j]>num[j+1]){
13.         temp=num[j];
14.         num[j]=num[j+1];
15.         num[j+1]=temp;
16.        }
17.      output+="\n\nArray contents in ascending order:\n";
18.      for(int i=0; i<num.length; i++)
19.       output+=num[i]+"    ";
20.     JOptionPane.showMessageDialog(null, output, title,
            JOptionPane.INFORMATION_MESSAGE);
21.      System.exit(0);
22.     }
23.    }
```

شرح المثال:

في الأسطر (١٠-١٦) هنالك عمليتا دوران متداخلتان يتم من خلالهما ترتيب عناصر المصفوفة num، حيث تم فحص شرط الترتيب في السطر رقم (١٢) وإذا تحقق هذا الشرط يتم تبديل عنصرين من عناصر المصفوفة بحيث يأخذ كل واحد من العنصرين مكان الآخر داخل المصفوفة. ومخرجات هذا البرنامج هي طباعة عناصر المصفوفة قبل الترتيب (من خلال الأسطر ٨-٩) وبعد الترتيب (من خلال الأسطر ١٨-١٩). هذا المثال يبين عملية الترتيب التصاعدي (Ascending). والشكل (١-١٢) يبين مخرجات البرنامج السابق:



شكل (١-١٢)

مثال ١-٨:

```
// array8.java

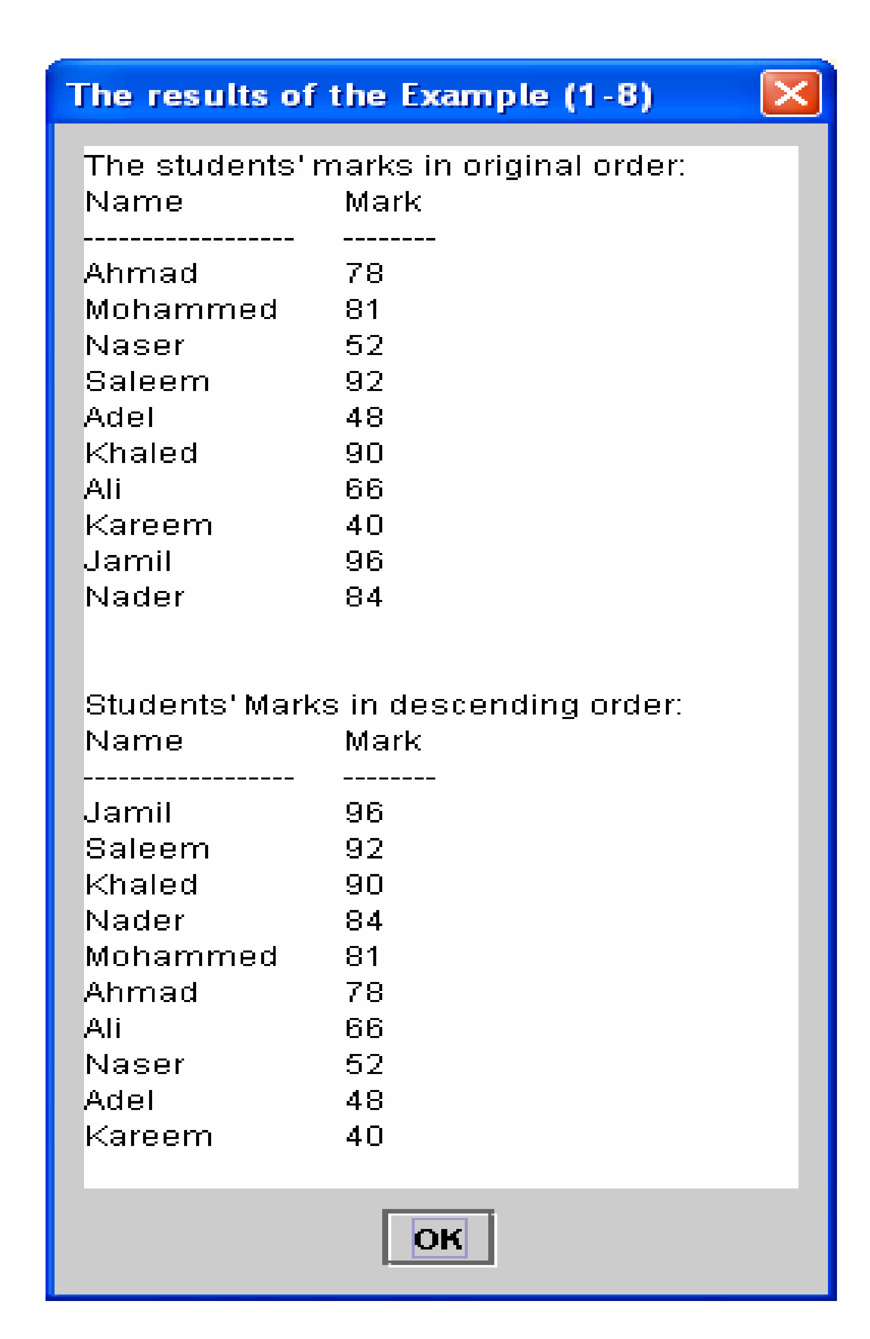
1.      import javax.swing.*;
2.      class array8{
3.       public static void main(String args[]){
4.        JTextArea outArea= new JTextArea();
5.           int mark[] = new int[] {78, 81, 52, 92, 48, 90, 66, 40, 96,84};
```

```
6.      String name[]= {"Ahmad", "Mohammed", "Naser", "Saleem",
                                "Adel", "Khaled", "Ali", "Kareem", "Jamil",
                                "Nader"};
7.      int temp_mark;
8.      String temp_name;
9.      String title="The results of the Example (1-8)";
10.     String output="The students' marks in original order:\n";
11.     output+="Name\tMark\n------------------\t--------\n";
12.     for(int i=0; i<mark.length; i++)
13.     output+=name[i]+"\t"+mark[i]+"\n";
14.     for(int i=1; i<mark.length; i++)
15.      for(int j=0; j<mark.length-1; j++)
16.       if(mark[j]<mark[j+1]){
17.        temp_mark=mark[j];
18.        mark[j]=mark[j+1];
19.        mark[j+1]=temp_mark;
20.        temp_name=name[j];
21.        name[j]=name[j+1];
22.        name[j+1]=temp_name;
23.       }
24.     output+="\n\nStudents' Marks in descending order:\n";
25.     output+="Name\tMark\n------------------\t--------\n";
26.     for(int i=0; i<mark.length; i++)
27.      output+=name[i]+"\t"+mark[i]+"\n";
28.     outArea.setText(output);
29.     JOptionPane.showMessageDialog(null, outArea, title,
            JOptionPane.PLAIN_MESSAGE);
30.     System.exit(0);
31.   }
32.  }
```

شرح المثال:

هذا المثال يبين عملية الترتيب التنازلي (Descending). في الأسطر (١٤-٢٣) تتم عملية ترتيب درجات الطلاب ترتيباً تنازلياً، حيث يتم من خلال الأسطر (١٧-١٩) تبديل الدرجات ليتم ترتيبها، وفي الأسطر

(٢٠-٢٢) يتم تبديل الأسماء لتبقى مرتبطة مع الدرجات الخاصة بها. والشكل (١-١٣) يبين مخرجات البرنامج السابق:



شكل (١-١٣)

البحث في المصفوفات (Searching):

عادةً يقوم المبرمج بالتعامل مع مصفوفات كبيرة الحجم وبالتالي لتحديد أي عنصر معين موجد في مصفوفة لابد من استخدام طرق البحث. من خلال هذا الدرس سوف نتعلّم طريقتين من طرق البحث وهما: البحث الخطي (linear Search) والبحث الثنائي (Binary Search).

يستخدم البحث الخطي للبحث عن عنصر معين داخل المصفوفة غير المرتبة أو المصفوفة المرتبة، وفي هذه الطريقة يتم مقارنة جميع محتويات المصفوفة مع القيمة المراد البحث عنها وبشكل متسلسل من بداية المصفوفة إلى نهايتها. بينما يستخدم البحث الثنائي للبحث عن عنصر معين داخل المصفوفة المرتبة فقط، وفي هذه الطريقة يتم تقسيم المصفوفة إلى نصفين وعن طريق المقارنة يتم تحديد إلى أي نصف ينتمي العنصر المراد البحث عنه وهكذا حتى يتم العثور على العنصر داخل المصفوفة إذا كان موجوداً. ويعتبر البحث الثنائي في المصفوفات المرتبة أسرع وأكفأ من البحث الخطي والأمثلة التالية توضح طرق البحث هذه.

مثال ١-٩:

```
// array9.java

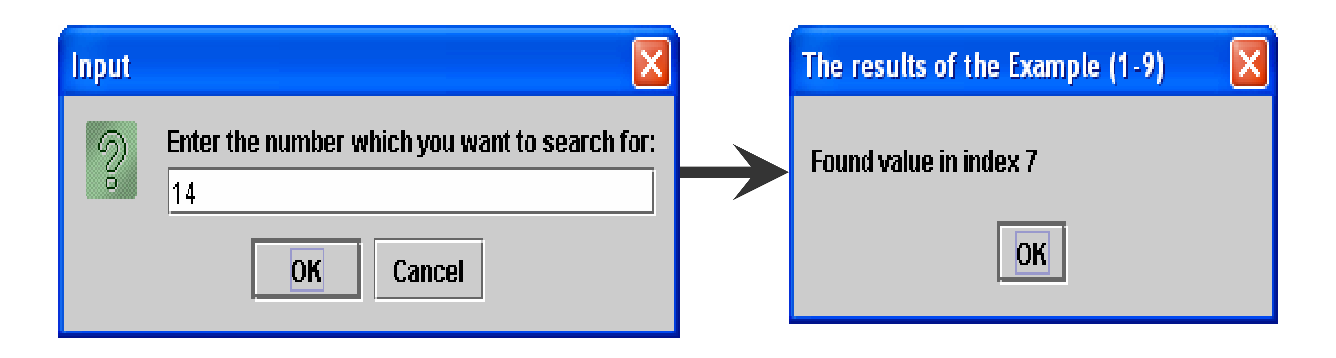
1.     import javax.swing.*;
2.     class array9{
3.      public static void main(String args[]){
4.       int n[] = new int[10];
5.       int num, k=-1;
6.       String title="The results of the Example (1-9)";
7.       String s, output="";

8.       for(int i=0; i<n.length; i++)
9.        n[i]=i*2;
10.    s=JOptionPane.showInputDialog("Enter the number which you
                                       want to search for:");
11.      num=Integer.parseInt(s);
12.      for(int i=1; i<n.length; i++)
13.       if(n[i]==num){
14.        k=i;
```

```
15.        break;
16.       }
17.     if(k!=-1)
18.      output+="Found value in index "+k;
19.     else
20.      output+="Value not found";
21.     JOptionPane.showMessageDialog(null, output, title,
                                                 JOptionPane.PLAIN_MESSAGE);
22.     System.exit(0);
23.    }
24.   }
```

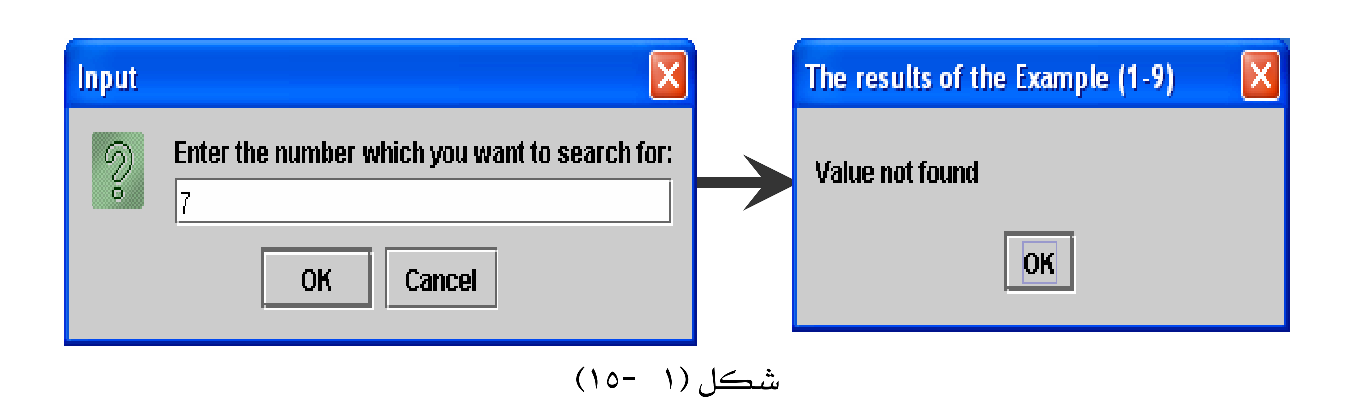
شرح المثال:

في هذا المثال تم استخدام طريقة البحث الخطي للبحث داخل مصفوفة مرتبة عن رقم معين يتم إدخاله عن طريق لوحة المفاتيح. حيث تقوم الأسطر (١٢-١٦) بمقارنة الرقم المراد البحث عنه والمخزن في المتغير num مع جميع محتويات المصفوفة، وفي حالة تم العثور على هذا الرقم في المصفوفة يتم تخزين موقعه في المتغير k، وفي حالة عدم العثور على الرقم في المصفوفة يبقى محتوى المتغير k كما هو ١-. ومن خلال الأسطر (١٧-٢٠) يتم طباعة رسالة بعدم وجود الرقم المراد البحث عنه في المصفوفة إذا كانت قيمة k لم تتغير وبقيت ١-، بينما إذا تغيرت قيمة k فهذا يعني بأن الرقم المراد البحث عنه موجود داخل المصفوفة وفي الموقع k وسوف يتم طباعة رسالة بذلك. والشكل (١-١٤) يبين تنفيذ البرنامج السابق في حالة العثور على الرقم ١٤ في المصفوفة:



شكل (١ -١٤)

بينما يبين الشكل (١-١٥) تنفيذ البرنامج في حالة عدم العثور على الرقم ٧ في المصفوفة:



شكل (١ -١٥)

مثال ١-١٠:

```
// array10.java

1.    import javax.swing.*;
2.    class array10{
3.     public static void main(String args[]){
4.      int id[] = new int[] {2, 10, 1, 7, 4, 6, 3, 8, 5, 9};
5.        String  name[]= {"Ahmad", "Mohammed", "Naser", "Saleem",
                          "Adel", "Khaled", "Ali", "Kareem", "Jamil",
                          "Nader"};

6.      String s, stdName, title="The results of the Example (1-10)";
7.      String output="The student's name is:--> ";
8.      int no, index=-1;
9.        s=JOptionPane.showInputDialog("Enter the student's ID to
                                         display his name:");
10.     no=Integer.parseInt(s);

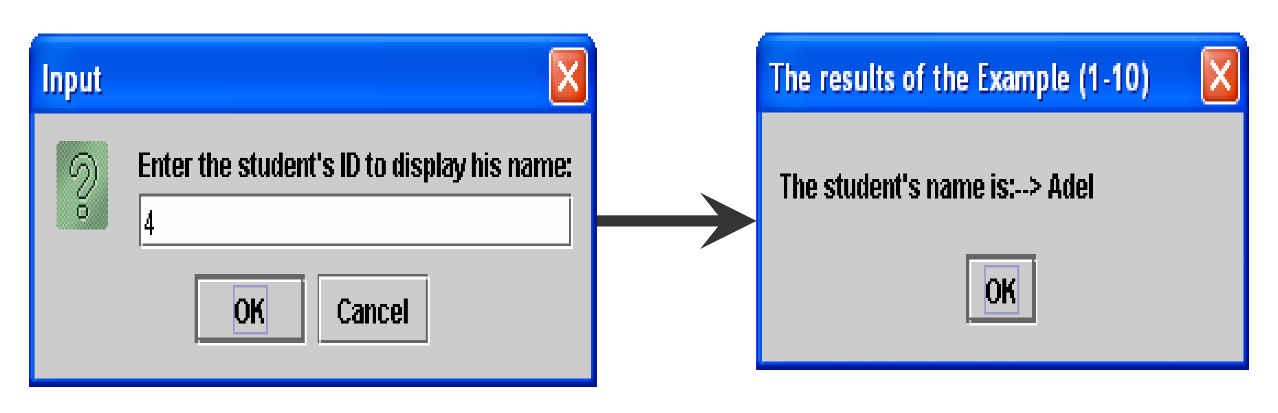
11.     for(int i=0; i<id.length; i++)
12.      if(id[i]==no){
13.       index=i;
14.       break;
15.      }
16.     if(index!=-1)
17.      output+=name[index];
```

```
18.     else
19.      output="There is no student with this ID !!!";
20.       JOptionPane.showMessageDialog(null, output, title,
                                    JOptionPane.PLAIN_MESSAGE);
21.     System.exit(0);
22.    }
23.   }
```

شرح المثال:

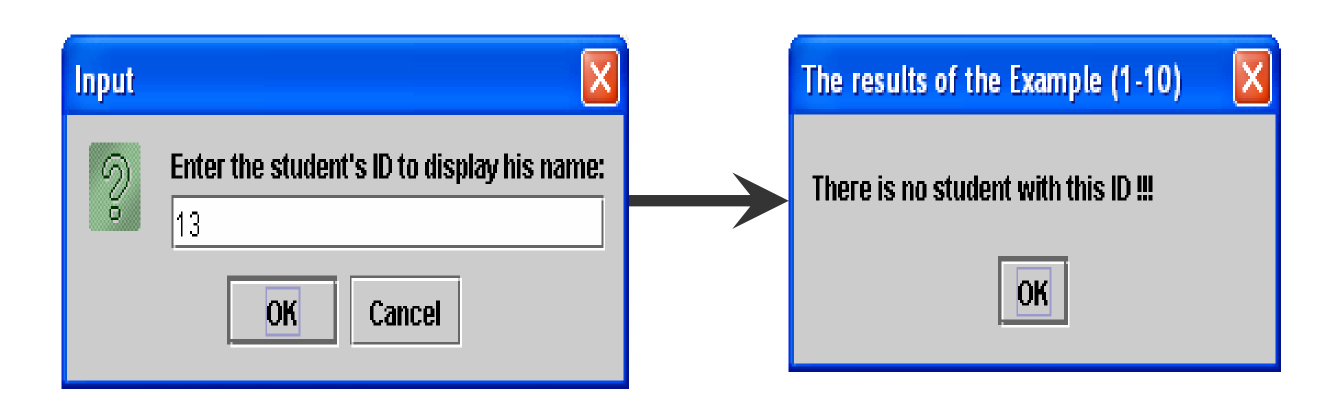
في هذا المثال تم استخدام طريقة البحث الخطي للبحث في مصفوفة غير مرتبة عن رقم طالب وطباعة اسم هذا الطالب. في الأسطر (١١-١٥) يتم البحث داخل المصفوفة id عن رقم الطالب الذي يتم إدخاله عن طريق لوحة المفاتيح (no)، فإذا كان هذا الرقم موجود في المصفوفة id يتم تخزين موقع رقم الطالب داخل المصفوفة id في المتغير index وبالتالي سوف تستخدم القيمة المخزنة في هذا المتغير لتحديد اسم الطالب صاحب هذا الرقم في المصفوفة name، حيث يتم من خلال الأسطر (١٦-١٩) إضافة اسم الطالب إلى المخرجات إذا كان رقم هذا الطالب موجوداً في المصفوفة id أو إضافة رسالة "There is no student with this ID !!!" إلى المخرجات إذا كان رقم الطالب المدخل غير موجود في هذه المصفوفة.

والشكل (١-١٦) يبين تنفيذ البرنامج السابق في حالة العثور على الطالب صاحب الرقم ٤.



شكل (١-١٦)

بينما يوضح الشكل (١-١٧) تنفيذ البرنامج السابق في حالة عدم العثور على أي طالب يحمل الرقم ١٣.



شكل (١-١٧)

مثال ١-١١:

```
// array11.java

1.      import javax.swing.*;
2.      class array11{
3.       public static void main(String args[]){
4.        int id[] = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
5.        String  name[]=  {"Ahmad",  "Mohammed",  "Naser",  "Saleem",
                            "Adel",  "Khaled",  "Ali",  "Kareem",  "Jamil",
                            "Nader"};

6.        String s, stdName, title="The results of the Example (1-11)";
7.        String output="The student's name is:--> ";
8.        int no, index=-1;
9.        int low=0;
10.       int high = id.length-1;
11.       int middle;
12.       s=JOptionPane.showInputDialog("Enter the student's ID to display
                                           his name:");
13.       no=Integer.parseInt(s);

14.       while(low <=high){

15.      middle=(low+high)/2;
16.       if(no==id[middle]){
```

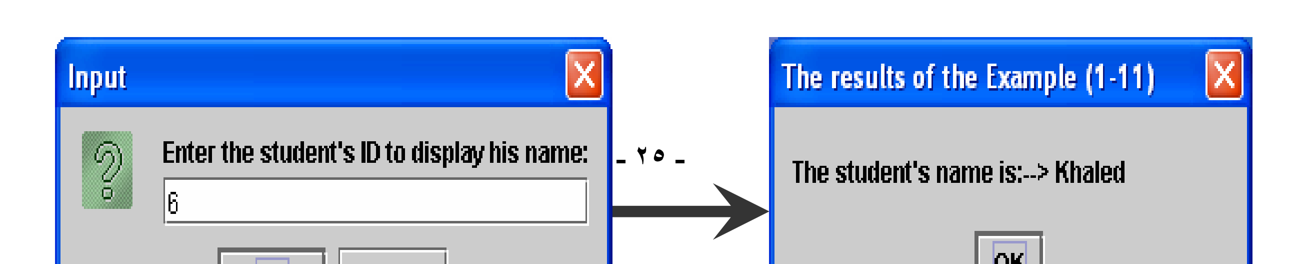
```
17.        index=middle;
18.        break;
19.       }
20.       else if(no<id[middle])
21.        high=middle-1;
22.       else
23.        low=middle+1;
24.     }

25.     if(index!=-1)
26.       output+=name[index];
27.     else
28.       output="There is no student with this ID !!!";
29.     JOptionPane.showMessageDialog(null, output, title,
                                          JOptionPane.PLAIN_MESSAGE)
30.     System.exit(0);
31.   }
32.  }
```

شرح المثال:

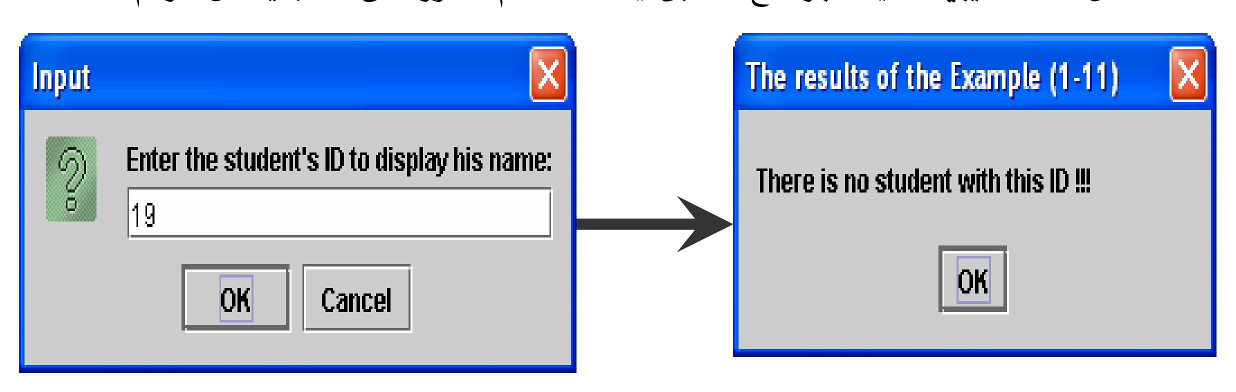
هذا المثال هو شبيه بالمثال السابق (١-١٠) حيث تم ترتيب أرقام الطلاب في المصفوفة id (السطر رقم ٤) واستخدمت طريقة البحث الثنائي والتي تعتبر أسرع وأكفأ عند البحث في المصفوفات المرتبة. في الأسطر (١٤-٢٤) تم تطبيق طريقة البحث الثنائي وذلك بتقسيم المصفوفة إلى نصفين وتحديد مكان وجود رقم الطالب المراد البحث عنه في أي نصف وبعد ذلك تقسيم النصف الذي ينتمي له رقم الطالب المراد البحث عنه إلى نصفين آخرين إلى أن يتم إيجاد رقم الطالب في المصفوفة id أو لغاية الخروج من الدوران (while) دون العثور على رقم الطالب في المصفوفة id.

والشكل (١-١٨) يبين تنفيذ البرنامج السابق في حالة العثور على الطالب صاحب الرقم ٦:



شكل (١-١٨)

بينما الشكل (١-١٩) يبين تنفيذ البرنامج السابق في حالة عدم العثور على طالب يحمل الرقم ١٩:



شكل (١-١٩)

مثال ١-١٢:

```
// array12.java

1.      public class array9{
2.       public static void main(String[] args) {
3.        int[] testArray = new int[50];
4.        testArray[43] = 10;
5.       int testArray2[] = { 35, 23, 8, 34, 66, 88, 5, 2, 85, 33 };
6.        int key, index=-1;
7.        key=10;
8.        System.out.println("Searching for element == 10");

9.        for(int i = 0; i < testArray.length; i++) {
10.        if(testArray[i] == key)
11.        index = i;
```

```
12.    }

13.    if(index != -1)
14.     System.out.println("Element found at " + index);
15.    else
16.     System.out.println("Element (10) does not found at the array
                                         testArray");
17.    index=-1;
18.    key=88;
19.    System.out.println("Searching the second array for element ==
                                         88");

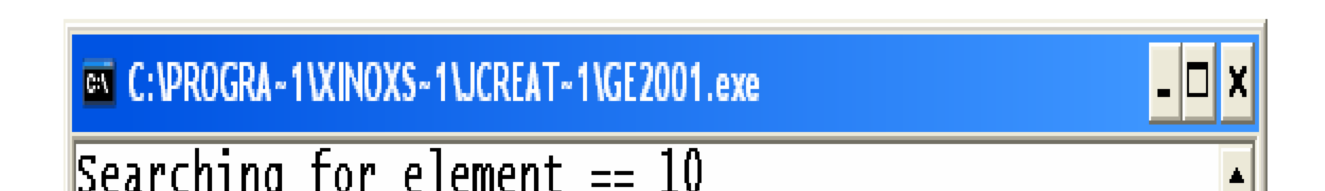
20.    for(int i = 0; i < testArray2.length; i++) {
21.     if(testArray2[i] == key)
22.     index = i;
23.    }

24.    if(index != -1)
25.     System.out.println("Element found at " + index);
26.    else
27.     System.out.println("Element (88) does not found at the array
                                             testArray2");
28.   }       }
```

شرح المثال:

وهذا المثال أيضاً يوضّح طريقة البحث الخطي، الأسطر (٩-١٢) والأسطر (٢٠-٢٣) توضّح عملية البحث الخطي عن طريق مقارنة العنصر المراد البحث عنه key مع جميع محتويات المصفوفة وبالترتيب، بحيث إذا وجد العنصر المراد البحث عنه تتم طباعة موقعه في المصفوفة وإذا لم يكن موجوداً في المصفوفة تتم طباعة رسالة بذلك.

والشكل (١-٢٠) يبين مخرجات البرنامج السابق.



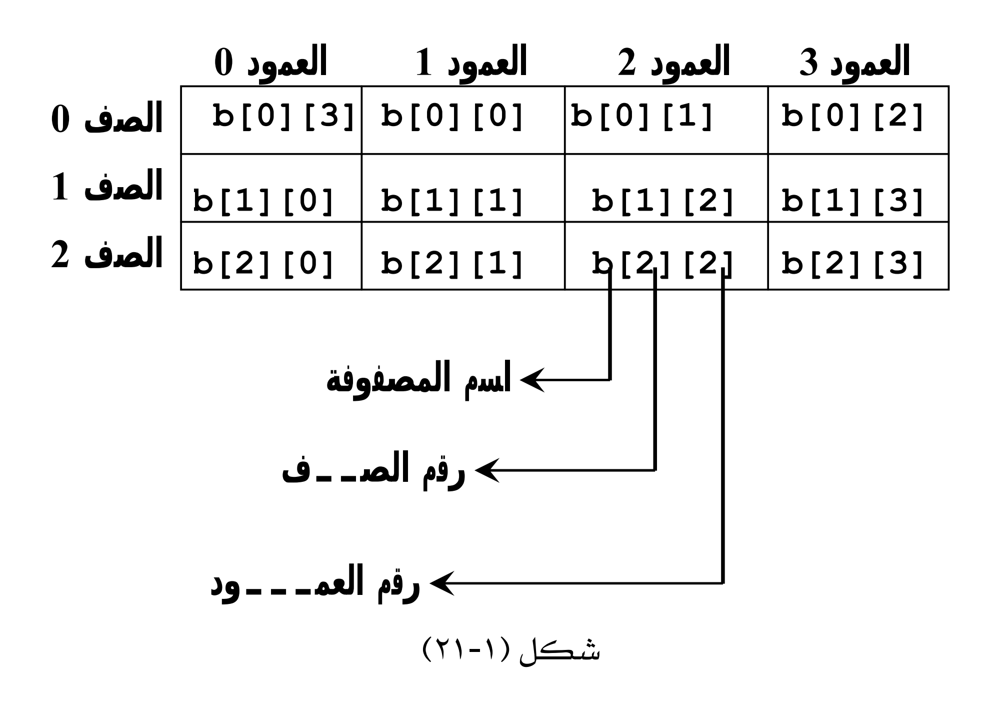
شكل (١-٢٠)

المصفوفات ذات البعدين (Two-Dimensional Arrays):

في لغة جافا يمكن تعريف مصفوفات ذات أكثر من بعد واحد، وكمثال على ذلك: تعريف المصفوفات ذات البعدين. ونستطيع القول بأن المصفوفة ذات البعدين هي عبارة عن جدول يحتوي على صفوف وأعمدة، انظر الشكل (١-٢١). والمثال التالي يوضع كيفية تعريف مصفوفة ذات بعدين وحجز مواقع لها:

```
1. int b[][];
2. b = new int[ 3 ][ 4 ];
```

في السطر الأول تم تعريف مصفوفة ذات بعدين، وفي الصف الثاني تم حجز مواقع لهذه المصفوفة بحيث تحتوي على ثلاث صفوف كل صف منها يحتوي على ثلاثة أعمدة. والشكل (١-٢٠) يوضح المصفوفة b وأرقام مواقعها.

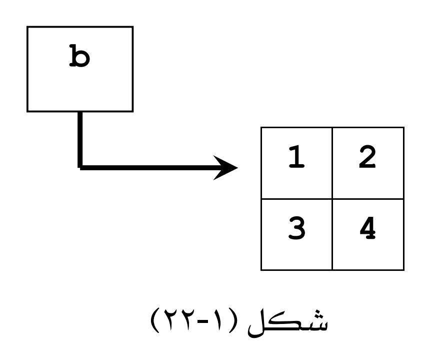


شكل (١-٢١)

والمثال التالي يوضح كيفية تعريف مصفوفة ذات بعدين وإعطائها قيماً ابتدائية:

```
int b[][] = { { 1, 2 }, { 3, 4 } };
```

في هذا المثال تم تخزين العدد ١ في المصفوفة b في الموقع الموجود في تقاطع الصف الأول والعمود الأول، والعدد ٢ في تقاطع الصف الأول والعمود الثاني. وبمعنى أخر تحتوي هذه المصفوفة على صفين، كل صف يحتوي على عنصرين. والشكل (١-٢٢) يبين محتويات المصفوفة b بعد تنفيذ الجملة السابقة:

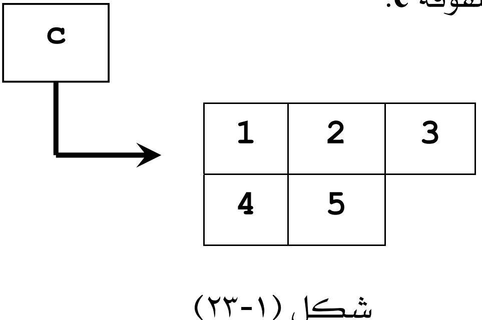


شكل (١-٢٢)

ويمكن لصفوف المصفوفة أن تحتوي على عدد مختلف من الأعمدة، بمعنى أن الصف الأول يحتوي على ثلاث أعمدة والصف الثاني يحتوي على عمودين فقط، كما في المثال التالي:

```
int c[][] = { { 1, 2, 3}, { 4, 5 } };
```

والشكل (١-٢٣) يبين محتويات المصفوفة **c**.

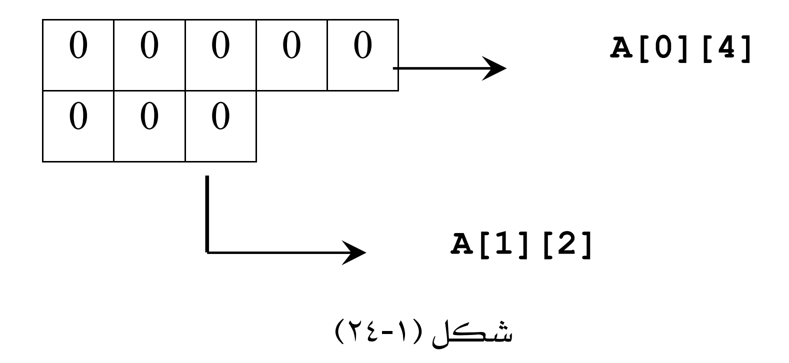


شكل (١-٢٣)

والمثال التالي يبين عملية تعريف مصفوفة وحجز مواقع لها بحيث يحتوي كل صف من صفوف هذه المصفوفة على عدد مختلف من الأعمدة:

```
1. int a[][];
2. a = new int[ 2 ][ ];   // allocate rows
3. a[ 0 ] = new int[ 5 ]; // allocate row 0
4. a[ 1 ] = new int[ 3 ]; // allocate row 1
```

في السطر رقم (١) تم تعريف مصفوفة اسمها a ، وفي السطر رقم (٢) تم حجز صفين لهذه المصفوفة ، بينما في السطر رقم (٣) تم حجز خمسة أعمدة للصف الأول ، ومن خلال السطر رقم (٤) تم حجز ثلاث أعمدة للصف الثاني. والشكل (١-٢٤) يوضح المصفوفة a.



شكل (١-٢٤)

أمثلة على المصفوفات ذات البعدين:

مثال ١-١٣:

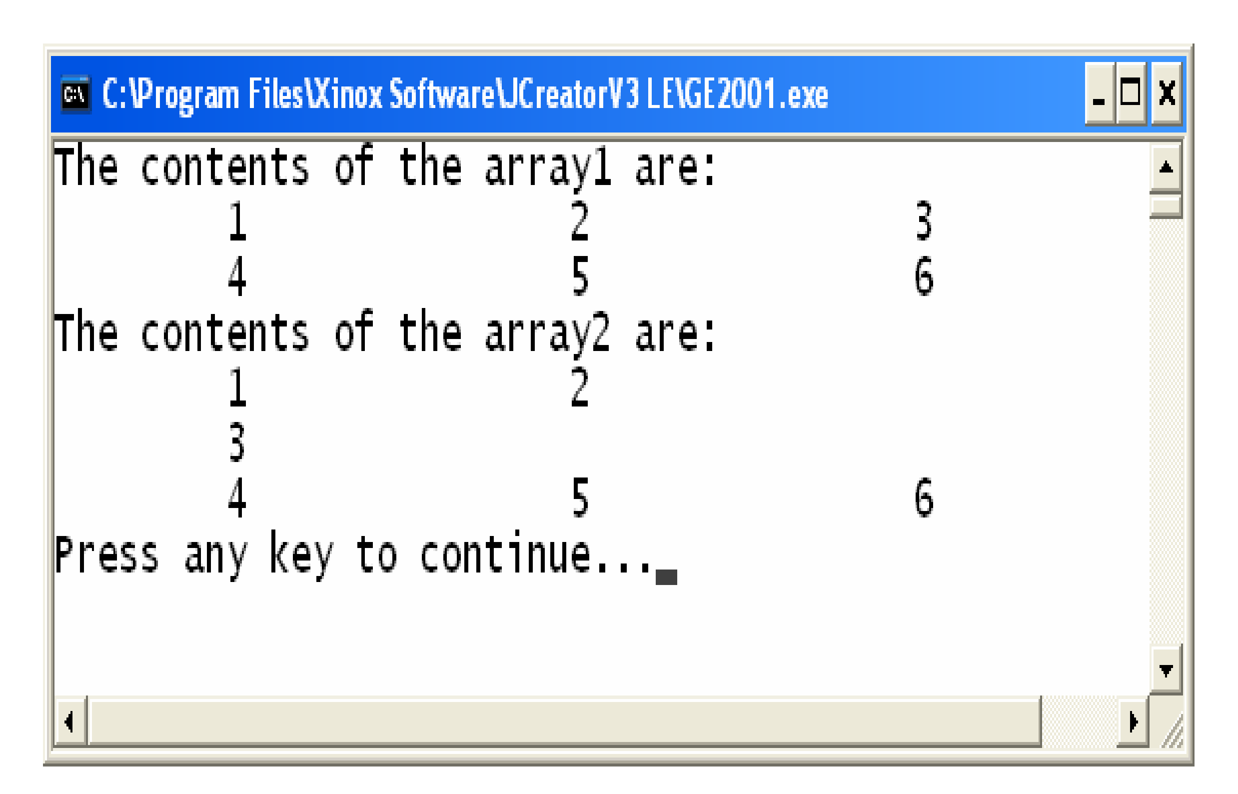
```
// array13.java

1.     public class array13 {
2.      public static void main(String[] args) {
3.       int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
4.       int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
5.       System.out.println("The contents of the array1 are:");
6.       for(int i=0; i<array1.length; i++){
7.        for(int j=0; j<array1[i].length; j++)
8.         System.out.print("\t"+array1[i][j]+"\t");
9.        System.out.println();
10.      }
11.      System.out.println("The contents of the array2 are:");
12.      for(int i=0; i<array2.length; i++){
13.       for(int j=0; j<array2[i].length; j++)
14.        System.out.print("\t"+array2[i][j]+"\t");
15.       System.out.println();
16.      }
17.     }
18.    }
```

شرح المثال:

في السطر رقم (٣) تم تعريف مصفوفة اسمها array1 وتم إعطاؤها قيماً ابتدائية، بحيث احتوت هذه المصفوفة على صفين وكل صف احتوى على ثلاثة أعمدة. وفي السطر رقم (٤) تم تعريف مصفوفة اسمها array2 وتم إعطاؤها قيماً ابتدائية، بحيث احتوت على ثلاث صفوف، الصف الأول فيها احتوى على عمودين، والصف الثاني احتوى على عمود واحد فقط، بينما الصف الثالث احتوى على ثلاث أعمدة. ومن خلال الأسطر (٦-١٠) تم طباعة محتويات المصفوفة array1 بحيث تكون المخرجات على شكل جدول، وقد تم معرفة عدد الصفوف في المصفوفة array1 من خلال الجملة array1.length وتم معرفة عدد

الأعمدة في كل صف من خلال الجملة array1[i].length حيث i يمثل رقم الصف. وتم طباعة محتويات المصفوفة array2 من خلال الأسطر (١٢-١٦). والشكل (١-٢٥) يبين مخرجات هذا البرنامج.



شكل (١-٢٥)

مثال ١-١٤:

```
// array14.java

1.      public class array14{

2.       public static void main(String[] args) {
3.        int grades[][] = { { 77, 68, 86, 73 },
4.                              { 96, 87, 89, 81 },
5.                              { 70, 90, 86, 81 } };

6.        int sum;
7.        System.out.println("The array is:");
```

```
8.      System.out.println("\t\t[0]\t[1]\t[2]\t[3]");

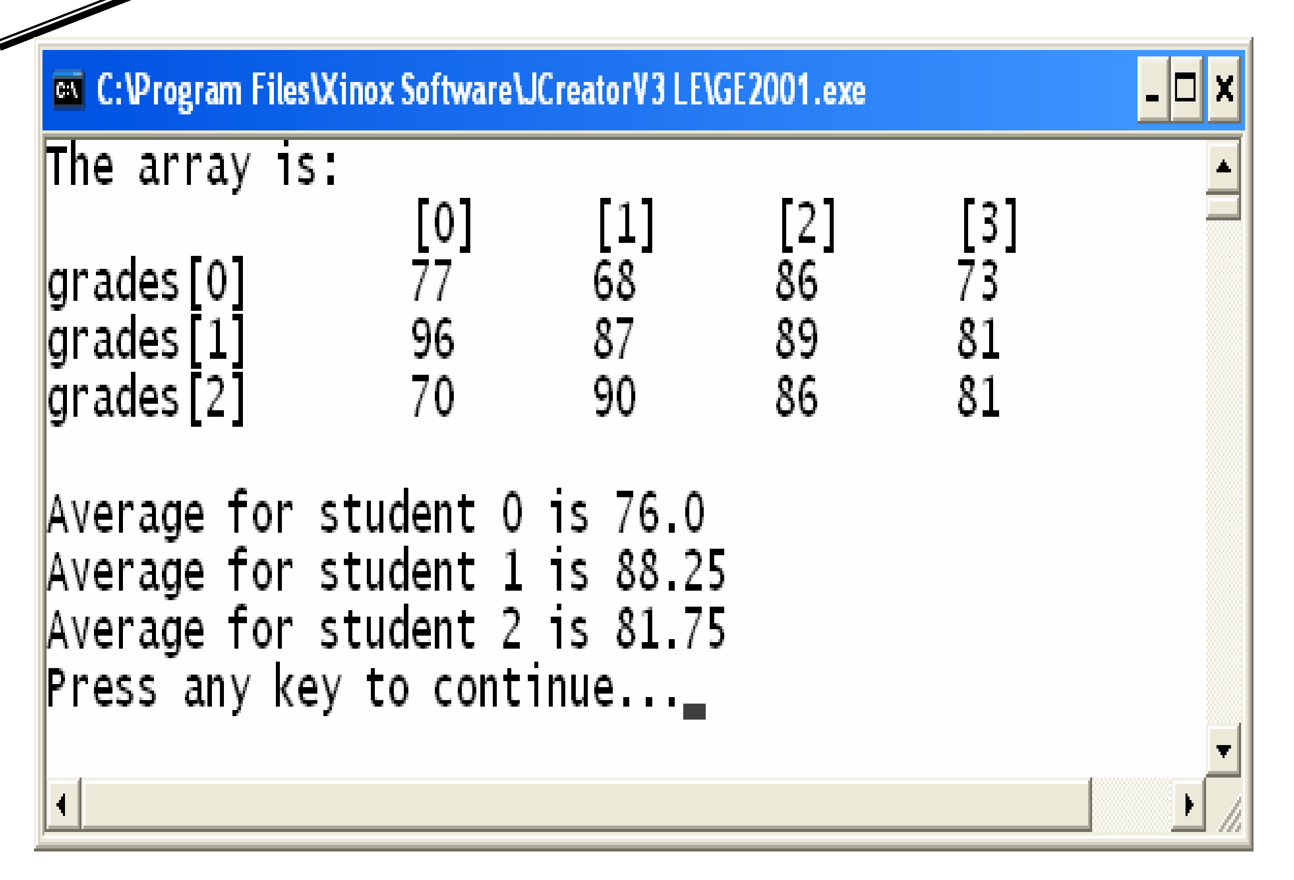
9.      for(int i=0; i<grades.length; i++){
10.      System.out.print("grades["+i+"]"+"\t");
11.      for(int j=0; j<grades[i].length; j++)
12.       System.out.print(grades[i][j]+"\t");
13.      System.out.println();
14.     }

15.     System.out.println();

16.     for(int i=0; i<grades.length; i++){
17.      sum=0;
18.      for(int j=0; j<grades[i].length; j++)
19.       sum+=grades[i][j];
20.      System.out.println("Average for student "+i+" is "
                                   +(double)sum/grades[i].length;
21.     }
22.    }
23.   }
```

شرح المثال:

هذا البرنامج يقوم بطباعة معدلات ثلاثة طلاب، كل طالب منهم له أربع درجات، حيث تم تعريف المصفوفة grades وتخزين الدرجات بها من خلال الأسطر (٣-٥). في الأسطر (٧-١٤) تم طباعة محتويات المصفوفة grades على شكل جدول مع توضيح أرقام الصفوف والأعمدة فيها. ومن خلال الأسطر (١٦-٢١) تم جمع درجات كل طالب لوحده (السطر رقم ١٩) ومن ثم إيجاد وطباعة المعدل لكل طالب (السطر رقم ٢٠)، حيث تم إيجاد المعدل لكل طالب وذلك بقسمة مجموع درجاته والمخزن في المتغير sum على عدد الدرجات والذي يمكن الحصول عليه لكل طالب من خلال الجملة grades[i].length حيث المتغير i يمثل رقم الصف (الطالب). يجب ملاحظة تصفير المتغير sum قبل جمع درجات كل طالب (السطر رقم ١٧). والشكل (١-٢٦) يبين مخرجات هذا البرنامج.



شكل (١-٢٦)

مثال ١-١٥:

```
// array15.java

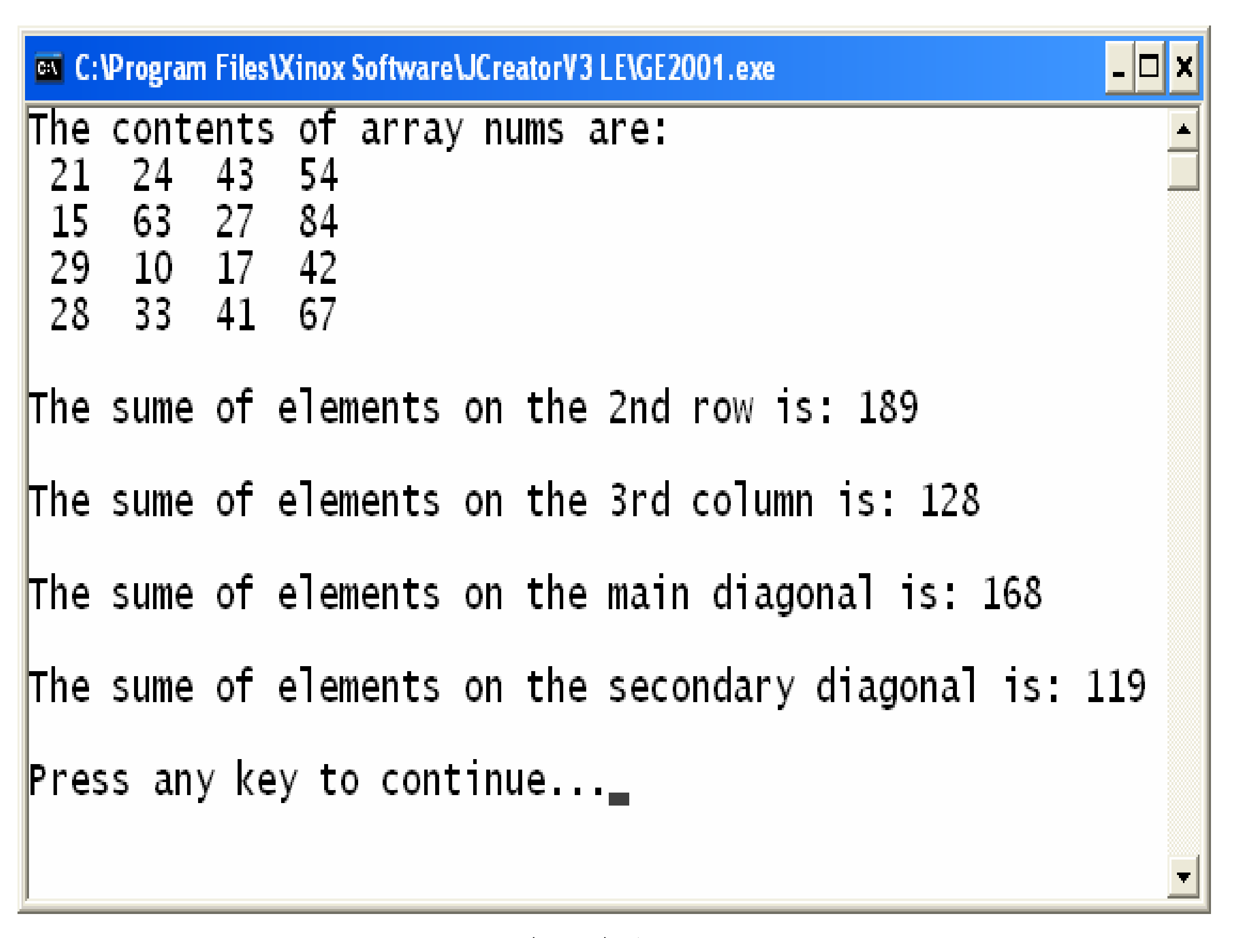
1.      public class array15{
2.       public static void main(String[] args) {
3.        int nums[][]= { {21, 24, 43, 54}, {15, 63, 27, 84}, {29, 10, 17, 42}, {28,
                                     33, 41, 67}
                                       };
4.        int sum=0;
5.        System.out.println("The contents of array nums are:");
6.        for(int i=0; i<nums.length; i++){
7.         for(int j=0; j<nums[i].length; j++)
8.          System.out.print(" "+nums[i][j]+" ");
9.         System.out.println();
10.       }
11.       for(int k=0; k<nums[1].length; k++)
```

```
12.      sum+=nums[1][k];
13.     System.out.println("\nThe sume of elements on the 2nd row is:
                                                 "+sum);
14.     sum=0;
15.     for(int k=0; k<nums.length; k++)
16.      sum+=nums[k][2];
17.     System.out.println("\nThe sume of elements on the 3rd column is:
                                                 "+sum);
18.     sum=0;
19.     for(int i=0; i<nums.length; i++)
20.      for(int j=0; j<nums[i].length; j++)
21.       if(i==j) sum+=nums[i][j];
22.        System.out.println("\nThe sume of elements on the main diagonal
                                                     is: "+sum);
23.     sum=0;
24.     for(int i=0; i<nums.length; i++)
25.      for(int j=0; j<nums[i].length; j++)
26.       if(i+j==nums.length-1) sum+=nums[i][j];
27.        System.out.println("\nThe sume of elements on the secondary
                                                 diagonal is: "+sum);
28.     System.out.println();
29.   }
30.  }
```

شرح المثال:

في هذا المثال تم التعامل مع صفوف وأعمدة معينه داخل المصفوفة، وتم التعامل مع القطر الرئيسي (ويمكن تحديده عندما يكون رقم الصف يساوي رقم العمود) والقطر الثانوي (ويمكن تحديده عندما يكون مجموع رقم الصف مع رقم العمود ناقص واحد يساوي عدد الصفوف (أو عدد الأعمدة) للمصفوفة، يجب ملاحظة أن القطر الرئيسي والقطر الثانوي يمكن التعامل معهم فقط في المصفوفات المربعة (المصفوفات المربعة هي المصفوفات التي يكون فيها عدد الصفوف وعدد الأعمدة متساويين). في الأسطر (٧-١١) تم طباعة محتويات المصفوفة على شكل جدول. في الأسطر (١٢-١٤) تم إيجاد وطباعة مجموع الأرقام المخزنة في الصف الثاني (رقم هذا الصف في المصفوفة هو ١). بينما تم في الأسطر (١٦-١٨) تم إيجاد وطباعة مجموع الأرقام المخزنة في العمود الثالث (رقم هذا العمود في المصفوفة هو ٢). وتم إيجاد وطباعة مجموع الأرقام الموجودة على القطر الرئيسي للمصفوفة nums من خلال الأسطر

(٢٠-٢٣)، لاحظ أنه باستخدام الشرط في السطر رقم (٢٢) تم معرفة فيما إذا كان هذا العنصر موجود على القطر الرئيسي أما لا. ومن خلال الأسطر (٢٥-٢٨) تم إيجاد وطباعة مجموع الأرقام الموجودة على القطر الثانوي للمصفوفة، لاحظ الشرط الذي من خلاله تم تحديد فيما إذا كان العنصر موجود على القطر الثانوي أم لا وذلك في السطر رقم (٢٧). والشكل (١-٢٧) يبين مخرجات هذا البرنامج.



شكل (١-٢٧)

تماريـــن:

س١: شركة تمنح موظفيها راتباً شهرياً مقداره ٢٥٠٠ ريال سعودي، وتمنح الشركة نسبة ٩٪ من مبيعات الموظف كعمولة تضاف إلى راتبه الشهري. اكتب برنامج يقرأ رواتب ١٠ موظفين ومجموع مبيعاتهم الشهرية بحيث تكون مخرجات البرنامج عبارة عن جدول يحتوي على عمولة الموظف وإجمالي راتب الموظف المكتسب في نهاية الشهر (إجمالي الراتب هو راتب الموظف ٢٥٠٠ ريال سعودي + ٩٪ من مجموع مبيعات الموظف في ذلك الشهر).

س٢: اكتب برنامج لقراءة ٢٠ عدد صحيح وتخزينها في مصفوفة ومن ثم فحص جميع الإعداد المخزنة في هذه المصفوفة وتخزين الأعداد الفردية في مصفوفة أخرى. وفي نهاية البرنامج اطبع محتويات المصفوفتين. (ملاحظة: يجب أن يكون حجم المصفوفة التي ستحوي الأعداد الفردية مساوياً لعدد هذه الأعداد).

س٣: اكتب برنامج لقراءة معدلات وأسماء ١٠ طلاب وتخزينهم في مصفوفتين (مصفوفة للمعدلات ومصفوفة للأسماء)، بحيث يقوم البرنامج بطباعة أسماء الطلاب الناجحين (اللذين تزيد معدلاتهم عن أو تساوي ٦٠) واسم الطالب صاحب أعلى درجة.

س٤: اكتب برنامج لقراءة N من الأعداد الحقيقة وتخزينها في مصفوفة، بحيث يقوم البرنامج بترتيب محتويات المصفوفة ترتيباً تصاعدياً. ويقوم البرنامج بقراءة عدد حقيقي من لوحة المفاتيح ليقوم بالبحث عن هذا العدد في المصفوفة بطريقة البحث الثنائي، فإذا وجد هذا العدد في المصفوفة يطبع البرنامج مكان وجود هذا العدد في المصفوفة وفي حال عدم وجوده يطبع "Not found in the array".

س٥: اكتب برنامج لتخزين أرقام وأسماء ورواتب موظفين في ثلاث مصفوفات. بحيث يستطيع المستخدم لهذا البرنامج البحث عن اسم وراتب موظف معين عن طريق رقمه (استخدم طريقة البحث الخطي). وكذلك يقوم البرنامج بطباعة أرقام وأسماء الموظفين اللذين تزيد رواتبهم عن ٢٥٠٠ ريال سعودي.

س٦: اكتب برنامج لتخزين جدول ضرب الخمسة في مصفوفة ذات بعدين، ومن ثم يقوم البرنامج بطباعة محتويات هذه المصفوفة.

س٧: لديك المصفوفة التالية:

٤	٦	٤	٩	٢
١	٣	٩	١	٦
٩	٩	٢	٥	٦
٣	٧	٤	٣	٨
٥	٣	٥	٦	٣

اكتب برنامج لطباعة ما يلي:

- مجموع الأعداد المخزنة في الصف الثاني والصف الرابع.
- الأعداد المخزنة في العمود الثالث.
- مجموع الأعداد المخزنة في القطر الرئيسي.
- معدل الأعداد المخزنة في القطر الثانوي.



برمجة ٢

الطـــــرق

الطـــرق

٢

الجـــــدارة:

معرفة كيفية كتابة الطرق بجميع أشكالها، ومعرفة كيفية استخدام الطرق الخاصة بالسلاسل الرمزية (String). بالإضافة للتعامل مع الطرق الموجودة في الصنف (Math).

الأهـــــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١- فهم ماهية الطرق (تعريفها واستخدامها).
٢- استخدام الطرق الموجودة مع الصنف (Math Class).
٣- معرفة فترة الحياة للمتغيرات (Life Time).
٤- معرفة مجال المتغيرات (Scope).
٥- معرفة استخدام الاستدعاء الذاتي (Recursion).
٦- معرفة واستخدام مفهوم (Overloading).
٧- التعامل مع الطرق الخاصة بالسلاسل الرمزية (String).

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ١٠ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

مقدمة:

في هذا الفصل سنقوم بالتعرف على كيفية تعريف الطرق وكيفية التعامل معها وأشكال استدعائها. وكذلك سنقوم باستخدام الطرق المتوفرة في بعض الأصناف الجاهزة والمتوفرة في مكتبة جافا، مثل تلك الطرق الخاصة بالتعامل مع السلاسل الرمزية والطرق الخاصة بالعمليات الحسابية، وفي نهاية هذه الوحدة هنالك عدد من التمارين.

ما هي الطرق (Methods)؟

الطريقة هي عبارة عن مجموعة من الجمل وتعرف بجسم الطريقة (Method Body) حيث يكون لها اسم معين، وتعرّف داخل الصنف. وتعرف الطريقة من خلال التوقيع (Signature) الخاص بها، وهو عبارة عن اسم الطريقة، نوع المعاملات وترتيبها، بالإضافة إلى نوع البيانات الراجعة منها.

والآن لنتعرف على عملية استخدام الطرق وذلك باستخدام تلك الطرق الموجودة في صنف العمليات الحسابية (Math Class).

صنف العمليات الحسابية (Math Class):

يحتوي هذا الصنف على العديد من الطرق التي تقوم بالعمليات الحسابية الشائعة مثل إيجاد القيمة المطلقة لعدد ،قوة العدد . . . الخ. وتتم عملية استدعاء الطرق بكتابة اسم الصنف متبوعا بنقطة بعدها اسم الطريقة ثم قائمة المعاملات داخل أقواس دائرية، كما يلي:

Class_Name.method_Name(*Argument List*)

مثال:

```
System.out.println(Math.sqrt(9.0)) ;
```

تقوم هذه الجملة باستدعاء الطريقة (sqrt) الموجودة في الصنف (Math) والتي تأخذ معامل واحد (9.0) من نوع (Double). فنتيجة تنفيذ هذه الجملة ستكون طباعة 3.0. والجدول (٢-١) يحتوي بعض الطرق الموجودة في الصنف (Math).

الطريقة	وصف الطريقة	مثال
abs(x)	القيمة المطلقة لـ **x**.	Math.abs(6.2) → 6.2 Math.abs(-2.4)→ 2.4
ceil(x)	تقرّب **x** إلى أقل عدد صحيح ليس اقل من **x**.	Math.ceil(5.1) → 6 Math.ceil(-5.1)→ -5
floor(x)	تقرب **x** إلى أكبر عدد صحيح ليس أكبر من **x**.	Math.floor(5.1) → 5 Math.floor(-5.1)→-6
max(x,y)	أكبر قيمة من **x** و **y**.	Math.max(7,6) → 7
min(x,y)	أقل قيمة من **x** و **y**.	Math.min(-7,-8) → -8
pow(x,y)	**x** مرفوعة للأس **y**.	Math.pow(6,2)→6^2 → 36
sqrt(x)	الجذر التربيعي لـ **x**.	Math.sqrt(9)→ $\sqrt{9}$→ 3
random()	تكوّن رقم عشوائي بين الصفر والواحد.	Math.random()→ 0.23121

جدول (٢-١)

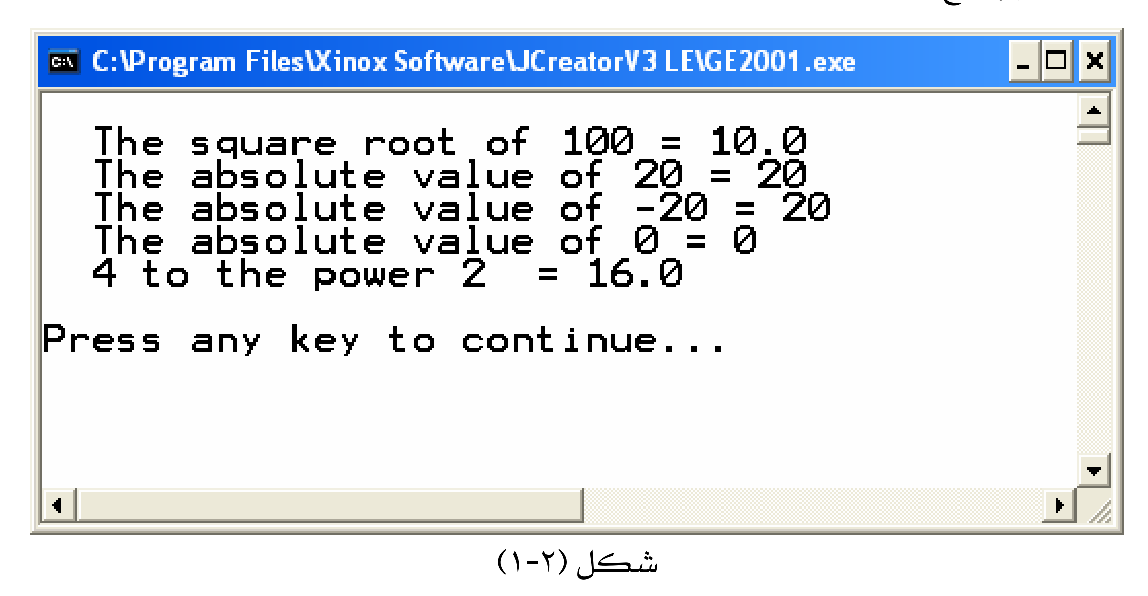
مثال ٢-١:

```
// UseMath.java

1.    public class UseMath{
2.     public static void main( String args[]){
3.      System.out.println("The square root of 100 = " + Math.sqrt(100));
4.      System.out.println("The absolute value of 20 = " + Math.abs(20));
5.      System.out.println("The absolute value of -20 = " + Math.abs(-20));
6.      System.out.println("The absolute value of 0 = " + Math.abs(0));
7.      System.out.println("4 to the power 2  = " + Math.pow(4,2));
8.     }      // end of main
9.    }       // end of class UseMath
```

شرح المثال:

من خلال المثال (٢-١) قمنا بالتعامل مع بعض الطرق الموجودة مع الصنف (Math)، والشكل (٢ - ١) يبين مخرجات هذا البرنامج.



شكل (٢-١)

وهنالك الكثير من المسائل التي تحتاج إلى استخدام الأرقام العشوائية مثل الألعاب وبرامج المحاكاة والمسابقات وغيرها. سنتعرف في المثال (٢-٢) على كيفية توليد الأرقام العشوائية واستخدامها من خلال مثال رمي حجر نرد ٥ مرات .

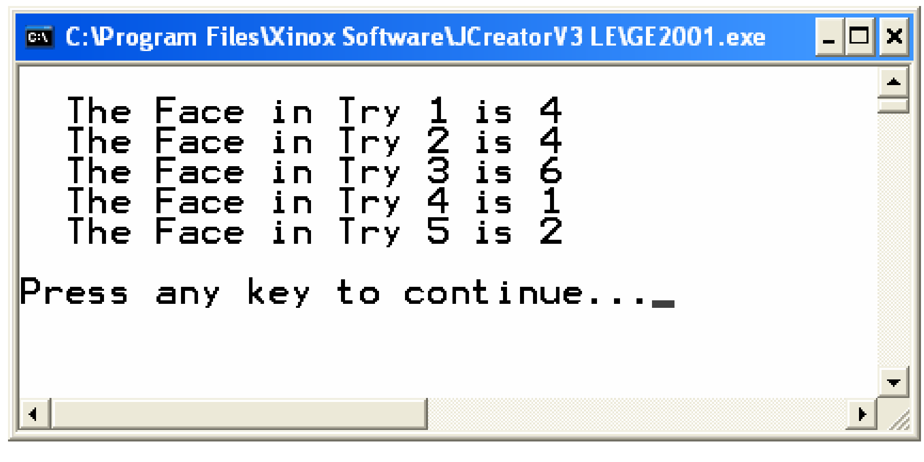
مثال (٢-٢):

```
// RollDie.java

1. public class RollDie{
2.  public static void main( String args[]){
3.   int face ;//variables to store the result
4.   for (int i = 1;i<=5;i++){
5.    face = 1+(int)(Math.random()*6);
6.  System.out.println("The Face in Try " + i + " is " + face);
7.    }  // end for loop
8.   } // end of main
9. }   // end of class RollDie
```

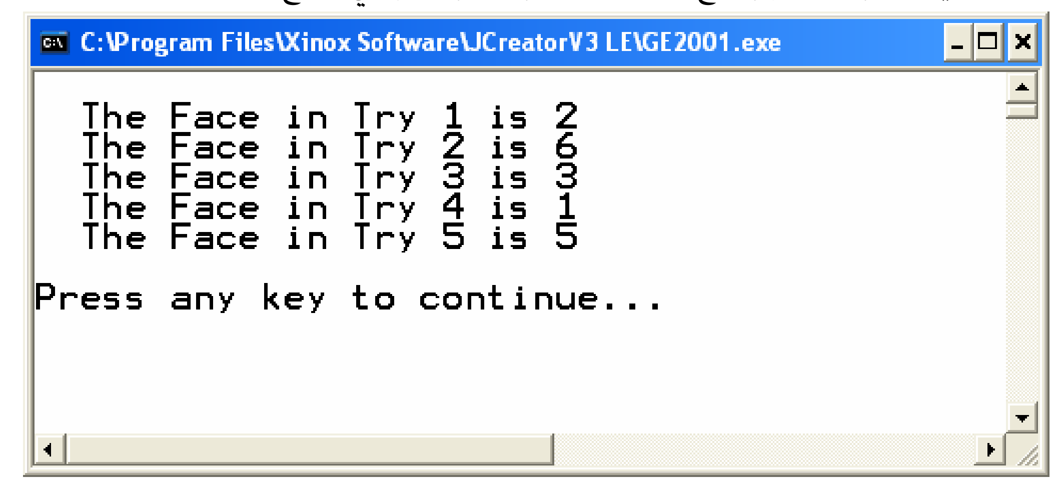
شرح المثال:

في هذا المثال يقوم البرنامج بتوليد ٥ أرقام عشوائية وذلك من خلال تنفيذ الجملة رقم (٥)، حيث تم استدعاء الطريقة random الموجودة في الصنف Math، وتقوم هذه الطريقة بتوليد رقم عشوائي أكبر من أو يساوي الصفر وأقل من واحد، ومن ثم تم ضرب الرقم العشوائي الناتج من (Math.random()) بالرقم ٦ وتحويله إلى عدد صحيح من خلال (int) ليصبح العدد العشوائي الناتج أكبر من أو يساوي صفر وأقل من أو يساوي خمسة، ومن ثم يضيف إلى الرقم العشوائي الرقم ١ ليصبح الرقم الناتج من تنفيذ السطر رقم (٦) اكبر من أو يساوي واحد وأقل من أو يساوي ستة. والشكل (٢-٢) يبين مخرجات هذا البرنامج. لاحظ أن نتائج هذا البرنامج قد تختلف في كل مرة ننفذ فيها البرنامج.



شكل (٢-٢)

والشكل (٢-٣) يبين مخرجات البرنامج بعد تنفيذه مرة أخرى، وهي نتائج مختلفة كما تلاحظ.



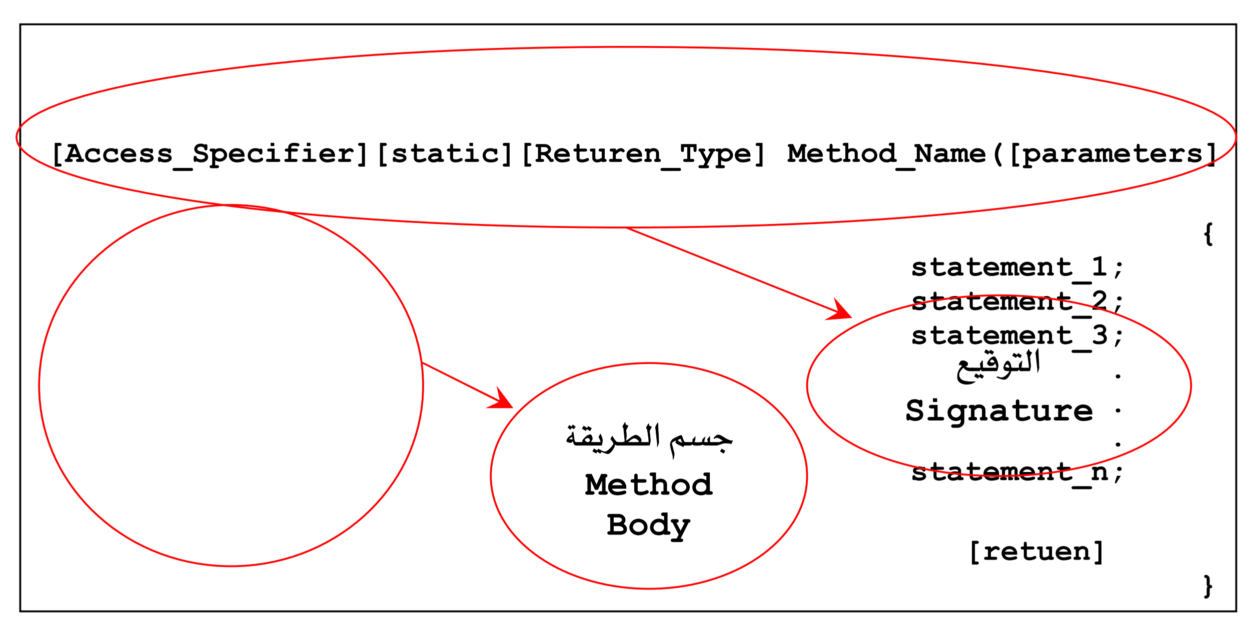
شكل (٢-٣)

فوائد استخدام الطرق:

ثبت عملياً أن أفضل طريقة لحل المسائل هي تقسيم هذه المسائل إلى وحدات صغيرة ويعرف هذا المبدأ بمبدأ "فرّق تسد" (Divide and Conquer) والطرق في جافا توفر لنا الإمكانية الكافية لتطبيق هذا المبدأ، مما سيسهل علينا كتابة البرنامج وتتبعه وإمكانية فهمه وصيانته بسهولة. وكذلك فإن استخدام الطرق المعرفة سابقا يوفر علينا كتابة البرنامج، وذلك من خلال إعادة استخدام هذه الطرق دون الحاجة إلى كتابتها مرة أخرى ودون الحاجة إلى معرفة ماهية هذه الطرق وكيف كتبت (Software Reusability)، ومثال ذلك استخدام تلك الطرق الموجودة في صنف العمليات الحسابية (Math Class)، والفائدة الأخرى هي تفادي تكرار كتابة الجمل في البرنامج، فما علينا سوى كتابة الجمل التي نحتاج إلى تكرارها في البرنامج داخل طريقة (Method)، ومن ثم نقوم باستدعاء هذه الطريقة عن طريق اسمها في أكثر من موقع في البرنامج ، كما سنرى لاحقاً.

تعريف الطرق واستدعائها:

كما ذكرنا سابقاً فالطريقة هي عبارة عن مجموعة من الجمل (وتعرف بجسم الطريقة (Method Body) حيث يكون لها اسم معين، وتعرّف داخل الصنف. وتعرّف الطريقة من خلال التوقيع (Signature) الخاص بها، وهو عبارة عن اسم الطريقة، نوع المعاملات وترتيبها ، بالإضافة إلى نوع البيانات الراجعة منها، والشكل (٢-٤) يبين الشكل العام لتعريف الطرق.



شكل (٢-٤)

وفي ما يلي شرح الشكل العام لتعريف الطرق، الأقواس المربعة "[" و "]" تدل على أن المحصور بينهما هو اختياري، أي يمكن أن يحذف من التوقيع الخاص للطريقة، لكن عند حذفها يجب مراعاة أمور أخرى سنتطرق لها لاحقاً في هذه الحقيبة إن شاء الله.

-(Access_Specifier) وهو محدد الوصول، ويمكن أن يكون واحد من المحددات التالية:

-(private): أي بمعنى "خاص"، بحيث تكون الطريقة (Method) مرئية فقط داخل الصنف (Class) الذي عرّفت فيه.

-(public): أي بمعنى "عام"، وتكون الطريقة مرئية في أي مكان في البرنامج.

-إذا لم يتم كتابة محدد الوصول (Access_Specifier) هذا يدل على أن هذه الطريقة مرئية في داخل الحزمة التي يتبع لها الصنف الذي عرفت الطريقة فيه.

وهنالك محددات وصول أخرى سوف يتم التطرق لها لاحقاً في الوحدة الثالثة من هذه الحقيبة.

-(static) اي بمعنى "ثابت"، وتستخدم لتعريف الطرق ليتم استخدامها داخل الصنف الذي عرّفت فيه فقط. (أي لا يمكن أن ترتبط بأي كائن (Object) من نوع هذا الصنف). وسوف نتطرق للكائن في الوحدة الثالثة، إن شاء الله.

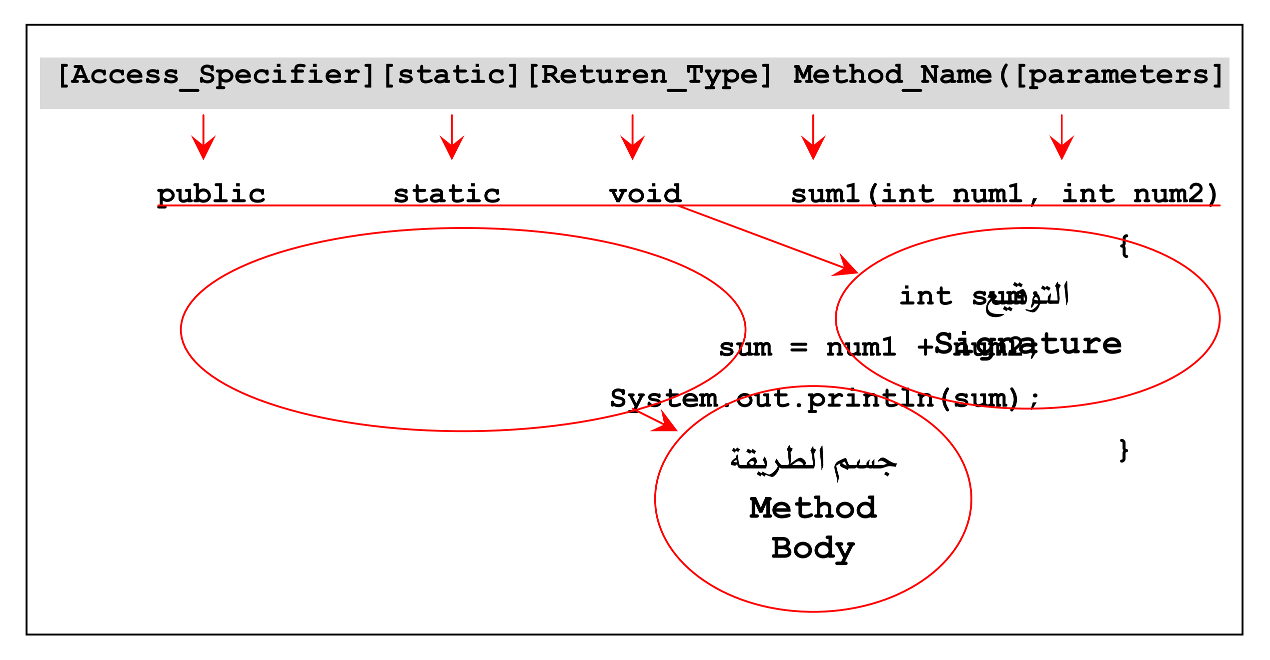
-(Return_Type) وهذا يحدد نوع البيانات التي ترجعها الطريقة عند استدعائها، ويمكن أن يكون نوع البيانات المرجعة أي نوع من أنواع البيانات (Data Types) التي تعرفها من خلال دراستك لمادة برمجة-١ (مثل:int، char، . . . الخ)، ويتم إرجاع القيمة باستخدام الكلمة المحجوزة (Return). ويمكن للطريقة أن لا ترجع أي قيمة، وفي هذه الحالة يجب أن يكون نوع البيانات المرجعة void.

-(Method_Name) وهو اسم الطريقة، ويجب مراعاة الشروط الخاصة بتحديد أسماء المتغيرات عند اختيار اسم للطريقة.

-(parameters) وهي المعاملات، وعند تعريف الطريقة تسمى هذه المعاملات بالمعاملات الشكلية (Formal Parameters)، ويمكن أن تستخدم هذه المعاملات في جسم الطريقة كمتغيرات بالإضافة للمتغيرات المحلية (Local Variables) التي تعرّف داخل جسم الطريقة. وعند استدعاء الطريقة تسمى المعاملات بالمعاملات الفعلية(Actual Parameters).

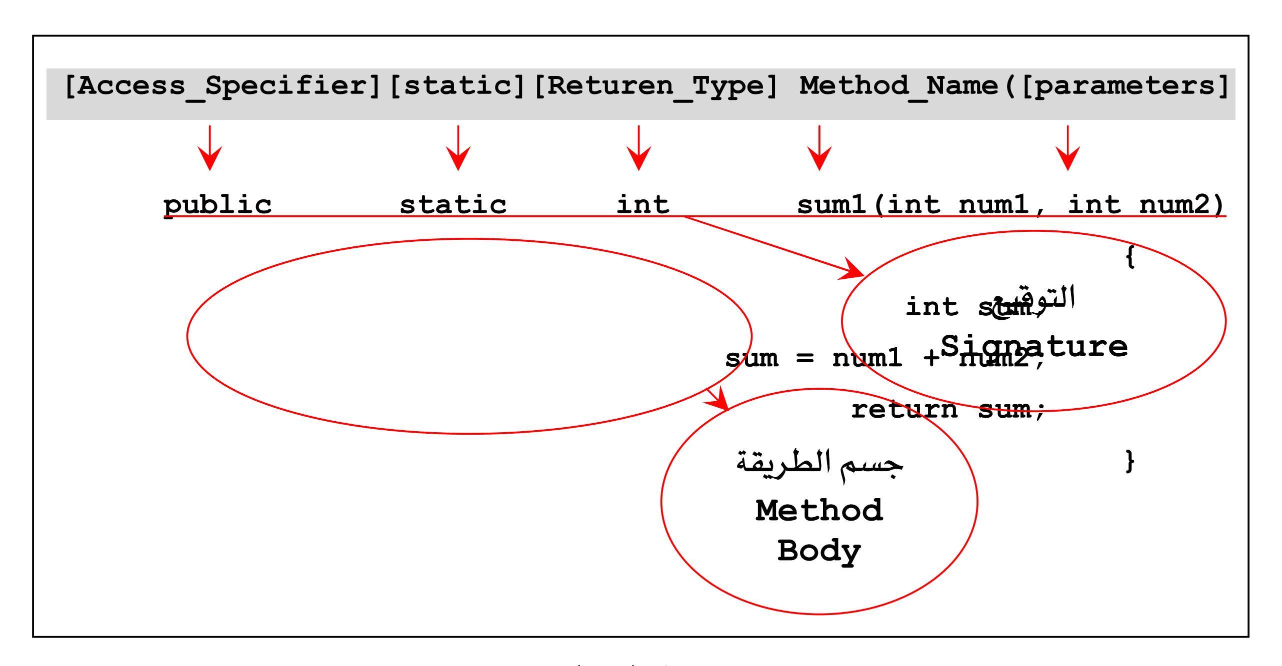
-(Method_Body) وهو جسم الطريقة، ويمكن أن يحتوي على تعريف المتغيرات المحلية والجمل التي يتم تنفيذها عند استدعاء هذه الطريقة. وإذا كان نوع البيانات المرجعة من هذه الطريقة غير النوع void فيجب أن يحتوي جسم الطريقة على جملة return التي توقف عمل الطريقة وترجع القيمة منها إلى مكان الاستدعاء.

والشكل (٢-٥) يبين كيفية تعريف طريقة لا ترجع أي قيمة (من نوع void)، بحيث تقوم هذه الطريقة والتي أسمها sum1 بجمع عددين وطباعة مجموعهما.



شكل (٢-٥)

والشكل (٢-٦) يبين كيفية كتابة الطريقة السابقة sum1 بحيث ترجع قيمة صحيحة وهي ناتج جمع العددين.



شكل (٢-٦)

ملاحظة: لا يجوز في لغة جافا كتابة طريقة داخل طريقة أخرى اطلاقاً.

مثال (٢-٣):

```
// Methods.java

1.      public class Methods {
2.      // instance variable declaration . . .
3.      public void method1(){
4.              //body
5.      }
6.      public void method2(int i , double j){
7.              //body
8.      }
9.
10.     public int method3(){
11.                     //body
12.     return 0; //integer expression
13.     }
14.
15.     public int method4(int i ,String s ){
16.                     //body
17.     return  0; //integer expression
18.     }
19.
20.     }
```

شرح المثال:

في المثال (٢-٣) تم تعريف أربع طرق لتوضيح الأشكال التي يمكن للطرق أن تأتي بها. خلال الاسطر (٣-٥) تم تعريف الطريقة (Method1) لا تأخذ معاملات ولا ترجع قيمة. والأسطر (٦-٨) تعرّف طريقة اسمها (Method2) لا ترجع قيمة لكن تأخذ المعاملات التالية: i من نوع int و j من نوع double. وفي الأسطر (١٠-١٣) تم تعريف الطريقة (Method3) والتي لا تأخذ معاملات لكن ترجع قيمة من نوع (int).

بينما في الاسطر (١٥-١٨) عرفت الطريقة (Method4) والتي تأخذ المعاملات التالية: i من نوع int و s من نوع String. وترجع قيمة من نوع (int).

تتم عملية استدعاء الطرق وبكل بساطة عن طريق كتابة اسم الطريقة وإرسال قيم المعاملات إن وجدت. ويتم ذلك في المكان المراد تنفيذ عمل الطريقة فيه، ويجب أن يكون الاستدعاء داخل طريقة أخرى. والشكل (٢-٧) يبين الشكل العام لعملية استدعاء الطرق.

```
Method_Name ( [Parameters_List] );
```

شكل (٢-٧)

وفي مايلي شرح الشكل العام لعملية استدعاء الطرق:

-(Method_Name): اسم الطريقة، وعند استدعاء طريقة موجودة في صنف آخر لابد من كتابة اسم هذا الصنف قبل اسم الطريقة بحيث تفصل بينهم نقطة.

-(Parameters_List): قائمة المعاملات الفعلية (Actual Parameters)، وهي القيم الفعلية التي تستخدم في عملية استدعاء الطرق، ويمكن أن تكون بالأشكال التالية:

- قيم ثابتة، مثل: sum1(5, 6).
- متغيرات، مثل: sum1(x, y).
- استدعاء لطريقة (Method) أخرى، مثل: sum1(sum2(z, 4), y).

وتأتي عملية استدعاء الطرق على شكلين هما:

١ -الطرق التي لا ترجع قيم (void) وفي هذه الحالة يجب أن لا يتم إسنادها إلى متغير ولا استخدامها في تعبير.

٢ -الطرق التي تقوم بإرجاع قيم وفي هذه الحالة يجب أن تستخدم في إحدى الحالات التالية :

- يتم إسنادها إلى متغير.
- استخدامها في تعبير.
- استخدامها في عملية استدعاء لطريقة أخرى، مثل: إرسالها للطريقة الخاصة بالطباعة System.out.println( ).

مثال (٢-٤):

```
// MethodCall.java

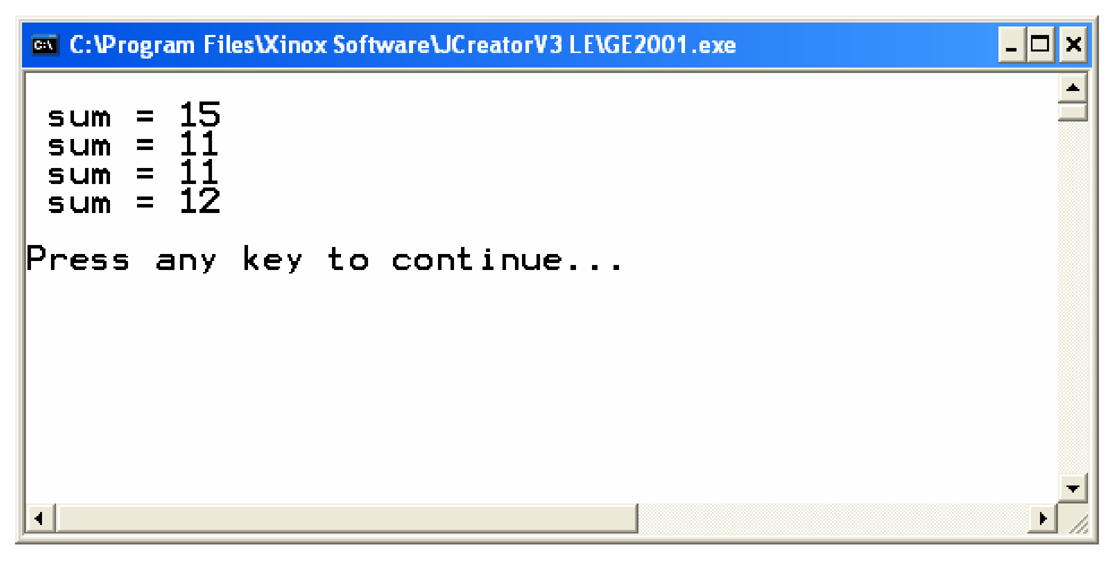
1.   public class MethodCall {
2.   public  static   void main(String args[]){
3.    int x = 5, y = 6, z = 0, s = 0;
4.    sum1(10, 5);
5.    sum1(x, y);
6.    s  =sum2(5, 6);
7.    System.out.println("sum = " +  sum2(5, 6));
8.    z = 12 + 3 * sum2(x, 10);
9.    sum1(sum2(3, 4), 5);
10.  } // end of  main
11.
12.  // defining the method sum1
13.  static void sum1(int num1,int num2){
14.    int sum=0;//local variable
15.    sum= num1+num2 ;
16.    System.out.println("sum = "+ sum);
17.  } // end of sum1
18.
19.  // defining the method sum2
20.  static int sum2(int num1,int num2){
21.    int sum=0; //  local variable
22.    sum= num1+num2 ;
23.    return sum ;// returned value
24.  }        // end of  sum2
25.  }// end of class MethodCall
```

شرح المثال:

في هذا المثال سوف نقوم بتوضيح عمليات الاستدعاء بشكليها السابقين. في السطر (٤) والسطر (٥) تم استدعاء الطريقة sum1 دون اسناد هذه الطريقة إلى أي متغير وذلك لأن هذه الطريقة لا ترجع أي قيمة

ونوع القيمة المرجعة فيها هو void، بحيث تم إرسال ثوابت في الاستدعاء الأول وتم إرسال متغيرات في الاستدعاء الثاني. بينما الطريقة sum2 ترجع قيمة من نوع int، في السطر (٦) تم إرسال ثوابت للطريقة sum2،وتم إسناد استدعاء هذه الطريقة إلى المتغير s ليتم تخزين القيمة المرجعة من هذه الطريقة في هذه المتغير. وفي السطر (٧) تم إرسال استدعاء الطريقة sum2 إلى الطريقة الخاصة بالطباعة وهي System.out.println(). والسطر (٨) يبين كيفية استدعاء الطريقة sum2 داخل تعبير حسابي. السطر (٩) يوضح عملية إرسال استدعاء الطريقة sum2 إلى الطريقة sum1.

والشكل (٢-٨) يبين نتائج تنفيذ البرنامج في المثال (٢-٤) السابق.



شكل (٢-٨)

ماذا يحدث عند استدعاء الطريقة (method)؟

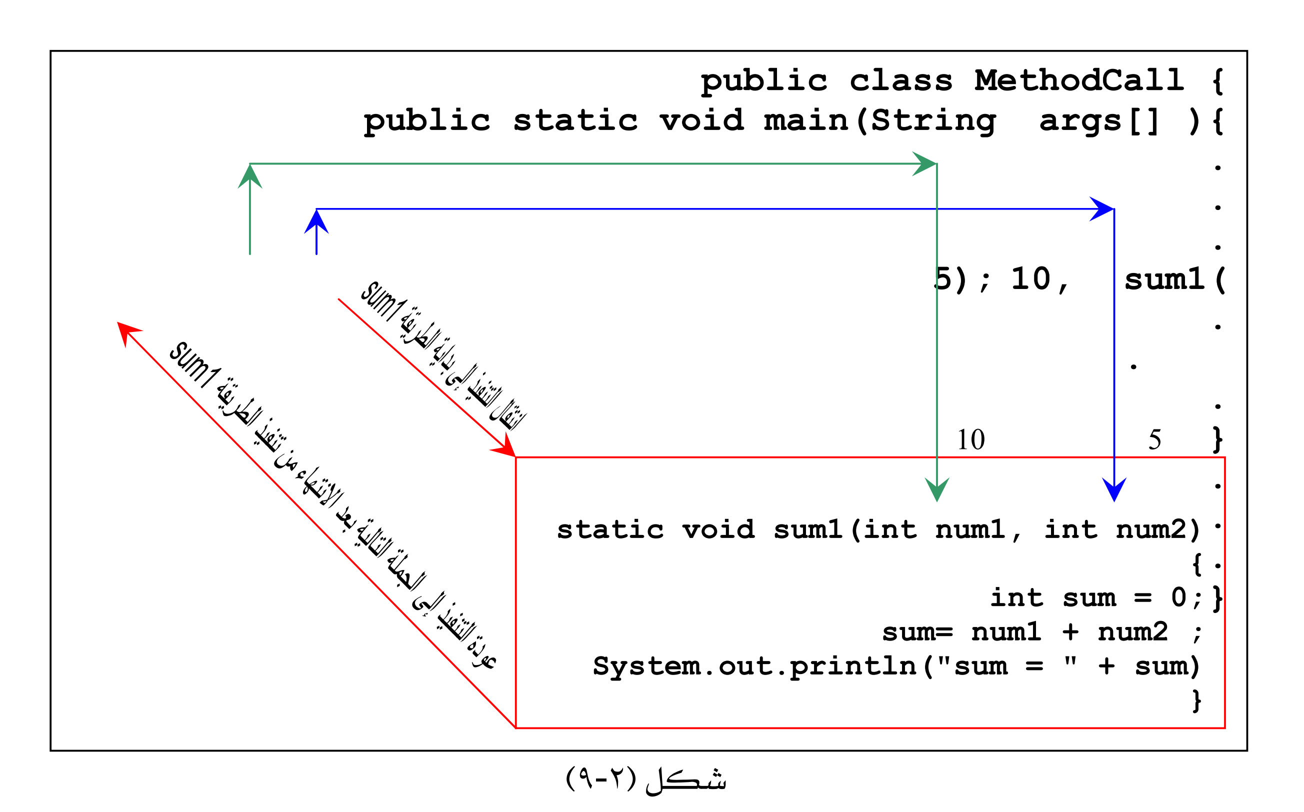
١- تنسخ المعاملات الفعلية (Actual parameters) إلى المعاملات الشكلية (parameters Formal) أي تصبح المعاملات الفعلية كقيم ابتدائية للمعاملات الشكلية. وتعمل المعاملات الشكلية عمل المتغيرات المحلية في جسم الطريقة.

٢- ينتقل تنفيذ البرنامج إلى بداية الطريقة المستدعاة .

٣- عند الانتهاء من تنفيذ الطريقة يستمر تنفيذ البرنامج من الجملة التالية لجملة الاستدعاء.

وكمثال على هذه الخطوات نعود إلى مثال (٢-٤) فمثلا عند استدعاء sum1(10,5) تنسخ القيمة ١٠ لِـ num1، والقيمة ٥ لِـ num2 ثم بعد ذلك ينتقل التنفيذ إلى بداية الطريقة sum1 ويستمر التنفيذ حتى نهاية

هذه الطريقة أو عند مواجهة جملة الرجوع (return). ثم بعد ذلك يعود التنفيذ إلى أول جملة بعد جملة الاستدعاء لِـ sum1 وهي الجملية الموجودة في السطر (٥). والشكل (٢-٩) يبين خطوات بدء تنفيذ الطريقة sum1 في السطر (٤) والعودة منها بعد الانتهاء من التنفيذ إلى السطر (٥).



شكل (٢-٩)

فترة حياة المتغيرات (Variable Life Time):

فترة الحياة للمتغير: هي الفترة التي يبقى فيها المتغير موجوداً داخل الذاكرة العشوائية (RAM)خلال تنفيذ البرنامج.

كما مر معنا سابقا فقد واجهنا أربعة أنواع من المتغيرات، فلنستعرض هذه الأنواع الأربعة من المتغيرات مع فترة حياتها بالنسبة للبرنامج :

١. المتغيرات الثابتة Static Variables: هي المتغيرات الخاصة بصنف أي أنها غير مرتبطة بأي كائن ينتمي لهذا الصنف. وتبدأ فترة حياة هذه المتغيرات عند عملية التحميل للصنف وتنهي عند إعادة التحميل لهذا الصنف.

٢. المتغيرات المحلية Local Variables: وهي المتغيرات المعرفة على مستوى المقطع (Block) الذي عرفت بداخله. أما فترة الحياة للمتغيرات المحلية فتبدأ عند إنشاء هذه المتغيرات وتنتهي عند الخروج من المقطع (block) الذي عرف به المتغير.

٣. المعاملات Parameter Variables: وهي التي تم تعرفيها في تعريف الطريقة (Method). وبالنسبة للمعاملات فتبدأ فترة الحياة عند استدعاء الطريقة (Method) وتنتهي عند الرجوع منها.

٤. متغيرات المثال Instance Variables: وهي المتغيرات الخاصة بالمثال (النسخة) المنشئ من صنف معين (وسوف تتم دراستها بالتفصيل في الفصل الخاص بالأصناف). وتبدأ فترة حياة متغيرات المثال عند إنشاء الكائن وتبقى مادامت أجزاء البرنامج تستطيع الوصول إلى هذا الكائن.

مجال المتغيرات (Variable Scope):

وهو الجزء من البرنامج الذي نستطيع من خلاله الوصول إلى المتغير. فبالنسبة لمتغيرات النسخة (Instance Variables) والطرق (Method) فنستطيع الوصول إليها داخل الصنف أي من بداية تعريف الصنف وحتى نهاية تعريفه. أما المتغيرات المحلية فإمكانية الوصول إليها تكون داخل المقطع (Block) الذي عرفت به فقط. اما بالنسبة للمتغيرات المحلية المعرفة على مستوى الطريقة والمعاملات فتكون إمكانية الوصول إليها داخل تلك الطريقة فقط .

مثال (٢-٥):

```
// VariableScope.java

1.     public class VariableScope{
2.      static int i;                //instance variable
3.      public static void main(String args[]){
4.       int x = 5, y = 6;         //local variables
5.       i = 10;
6.       System.out.println("i = " + i);
7.       i = method1(x, y);
8.       System.out.println("i = " + i);
9.       i = method2(x, y);
10.      System.out.println("i = " + i);
11.    }      //end main
```

```
12.
13.     static int method1(int arg11 ,int arg12 ){
14.      double num11 ,num12;
15.      for(int counter = 0; counter <= 5; counter++){
16.       i+= counter;
17.       }    //end of for counter loop
18.      return i+arg11+arg12;
19.     }      //end method1
20.
21.     static int  method2 (int  arg21, int arg22){
22.      int num21, num22, i=0;   //local variables
23.      {
24.       String s; //local variable
25.      }
26.      return i+arg21+arg22;
27.     }      //end method1
28.   }             //end of class VariableScope
```

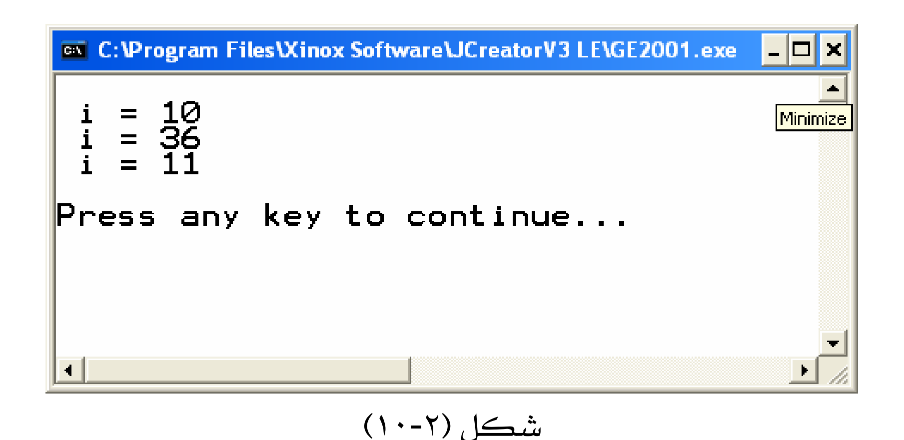
شرح المثال:

من خلال هذا المثال سوف نتعرف على أن مجال المتغير (Variable Scope) يؤثر على الممكان الممكن استخدام هذا المتغير فيه. في السطر (٢) تم تعريف المتغير i ليكون مرئي على مستوى الصنف VariableScope كاملاً، حيث تكون فترة حياة هذا المتغير من بداية تحميل الصنف إلى نهايته، وبما أن هذا الصنف يحتوي على الطريقة ()main فإنه يعتبر الصنف الرئيسي لتنفيذ البرنامج، وبذلك تكون فترة حياة المتغير i من بداية البرنامج إلى نهايته). في السطر (٤) تم تعريف المتغيرين x و y كمتغيرات محلية (Local Variables) يمكن رؤيتها داخل الطريقة ()main فقط، وفترة حياتهما تمتد من بداية الطريقة ()main إلى نهايتها. في السطر (١٣) المعاملان arg11 و arg12 الخاصين بالطريقة method1 فيكونان مرئيان فقط داخل هذه الطريقة، وفترة حياتهما تبدأ من لحظة استدعاء الطريقة ولغاية الانتهاء من هذه الطريقة والخروج منها. في السطر (١٤) المتغيران num11 و num12 هما متغيران محليان ويكونان مرئيان داخل الطريقة method1 فقط، وتبدأ فترة حياتهما باستدعاء الطريقة وتنتهي بالخروج منها. في السطر (١٥) تم تعريف المتغير counter ليكون مرئي داخل جملة الدوران for فقط، وتمتد فترة حياة هذا المتغير من لحظة الدخول إلى جملة الدوران وتستمر حتى نهاية المقطع (block) الخاص بهذه الجملة. وفي السطر

(٢٢) تم تعريف متغيرات محلية للطريقة method2 ومن هذه المتغيرات متغير اسمه i ،ونلاحظ أن اسم هذا المتغير يتطابق مع اسم المتغير المعرّف على مستوى الصنف في السطر (٢)، وهذا التعريف يلغي رؤية المتغير i المعرّف على مستوى الصنف داخل هذه الطريقة. وعند استخدام المتغير داخل الطريقة فهذا يعني الرجوع للمتغير المعرّف على مستوى الطريقة فقط. وفي السطر (٢٤) تم تعريف المتغير s على مستوى المقطع (Block) الذي يبدأ من السطر (٢٣) وينتهي بالسطر (٢٥)، وبهذا يكون مجال رؤية هذا المتغير داخل هذا المقطع فقط وفترة حياته تبدأ من بداية المقطع وتنتهي بنهاية المقطع.

والآن سوف نشرح تنفيذ بعض جمل البرنامج، في السطر (٥) يتم اسناد الرقم ١٠ إلى المتغير i المعرّف على مستوى الصنف.، وبعد ذلك وفي السطر (٦) يتم طباعة محتويات المتغير i. وفي السطر (٧) يتم استدعاء الطريقة Method1 و إرسال ٥ و ٦ إلى معاملاتها x و y وبالترتيب.، وبعد ذلك ترجع هذه الطريقة ٣٦ ليتم تخزين هذا الرقم في i. ومن ثم يتم طباعة محتويات i من خلال تنفيذ الجملة في السطر (٨). وفي السطر (٩) يتم استدعاء الطريقة method2 بإرسال ٥ و ٦ لمعاملتها، بحيث ترجع هذه الطريقة ١١ ليتم تخزينه في i، ومن خلال السطر (١٠) يتم طباعة محتويات i. لاحظ أن قيمة i داخل الطريقة method2 هي صفر.

والشكل (٢-١٠) يبين نتائج تنفيذ هذا البرنامج.



شكل (٢-١٠)

انواع تمرير البيانات:

هنالك طريقتان لعملية تمرير البيانات إلى الطرق:

١ -التمرير باستخدام القيمة (Pass-By-Value): وفي هذا النوع يتم إرسال نسخة من قيمة المتغير (المعامل الفعلي) إلى معامل الطريقة (المعامل الشكلي) المقابل له. أي أن عملية التعديل على المعامل الشكلي لا تؤثر على المتغير (المعامل الفعلي) الذي تم إرساله إلى الطريقة عند الاستدعاء. حيث أن

هذا النوع من تمرير البيانات يتم تطبيقه تلقائياً عندما يكون نوع المعاملات الفعلية من الأنواع البدائية (Primitive Data Types) مثل: int، double، float، . . . الخ.

٢ -التمرير باستخدام العنوان (Pass-By-Reference): وفي هذا النوع يتم إرسال عنوان المتغير (المعامل الفعلي) إلى المعامل الشكلي المقابل له في الطريقة (Method)، ليصبح المعامل الشكلي والمعامل الفعلي يؤشران إلى نفس العنوان في الذاكرة الرئيسية. وفي هذا النوع أي تغيير يتم على المعامل الشكلي يؤثر وفي نفس الوقت على المعامل الفعلي الذي تم إرساله للطريقة عند الاستدعاء.وهذا النوع من تمرير البيانات يتم تطبيقه بشكل تلقائي عندما تكون المعاملات الفعلية من أحد الكائنات (Objects) مثل: (المصفوفات) Arrays، String، . . . الخ.

مثال (٢-٦):

```
// Passing_Parameters.java

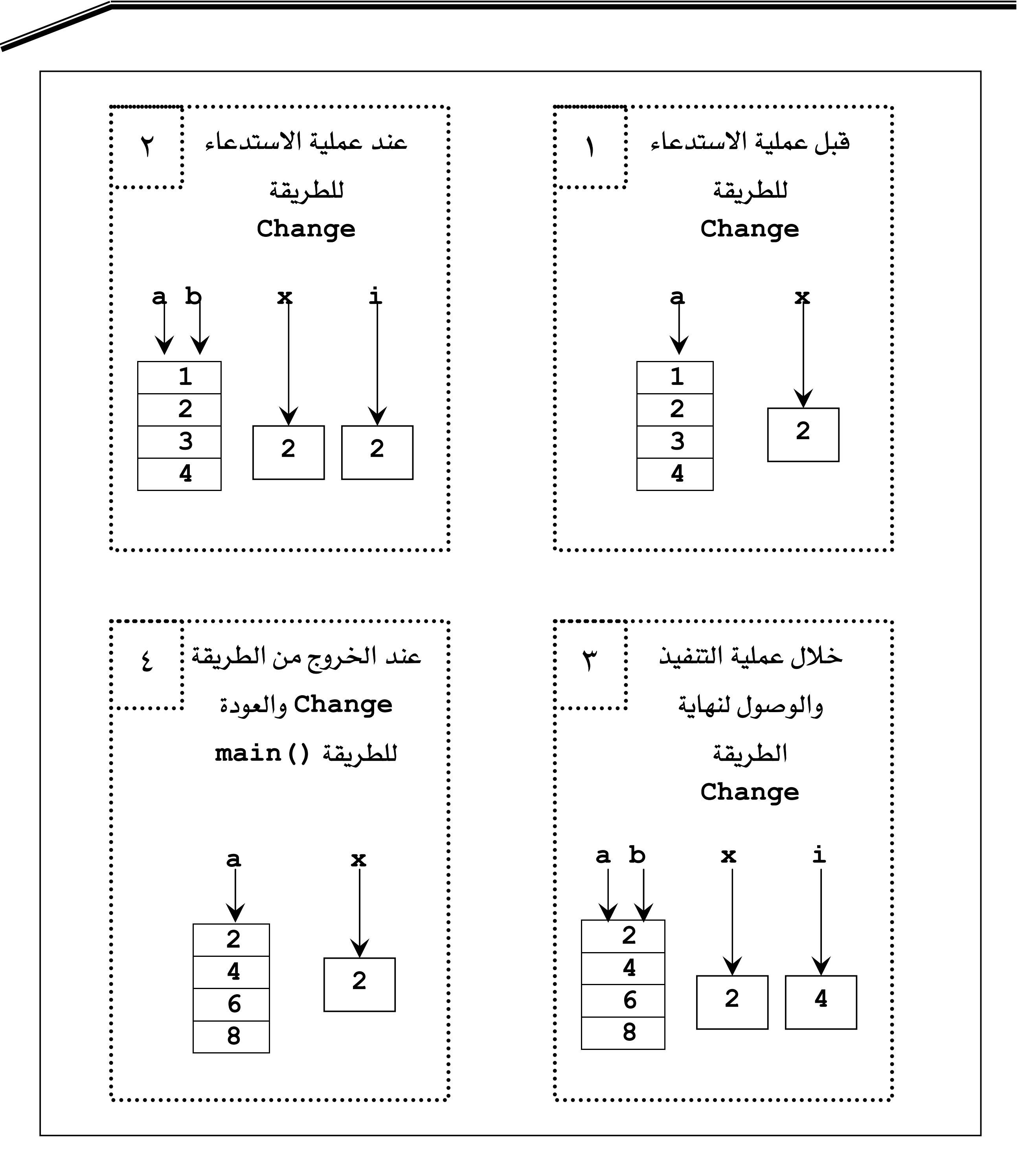
1. public class Passing_Parametres{
2.  public static void main(String args[]){
3.  int x;
4.  int a[] = {1, 2, 3, 4};
5.  x = a[1];
6.  System.out.println("The value of x before change is" + x) ;
7.  System.out.println("The value of a elements before change is: ");
8.  printArray(a);
9.  change(a, x);
10.     System.out.println("The value of x after change is" + x);
11.     System.out.println("The value of a elements after change is: ");
12.     printArray(a);


13.  } //end of main
14.   static void change(int b[], int i){
15.       i *= 2 ;
16.     for (int index=0; index < b.length; index++)
17.     b[index]*= 2;
18.  }       //end of method change
```

```
19. static void printArray(int c[]){
20.    for (int index=0; index < c.length; index++)
21.    System.out.print(c[index] + "\t");
22.    System.out.println();
23. }       //end of method print
24. }       //end of class Passing_Parametres
```

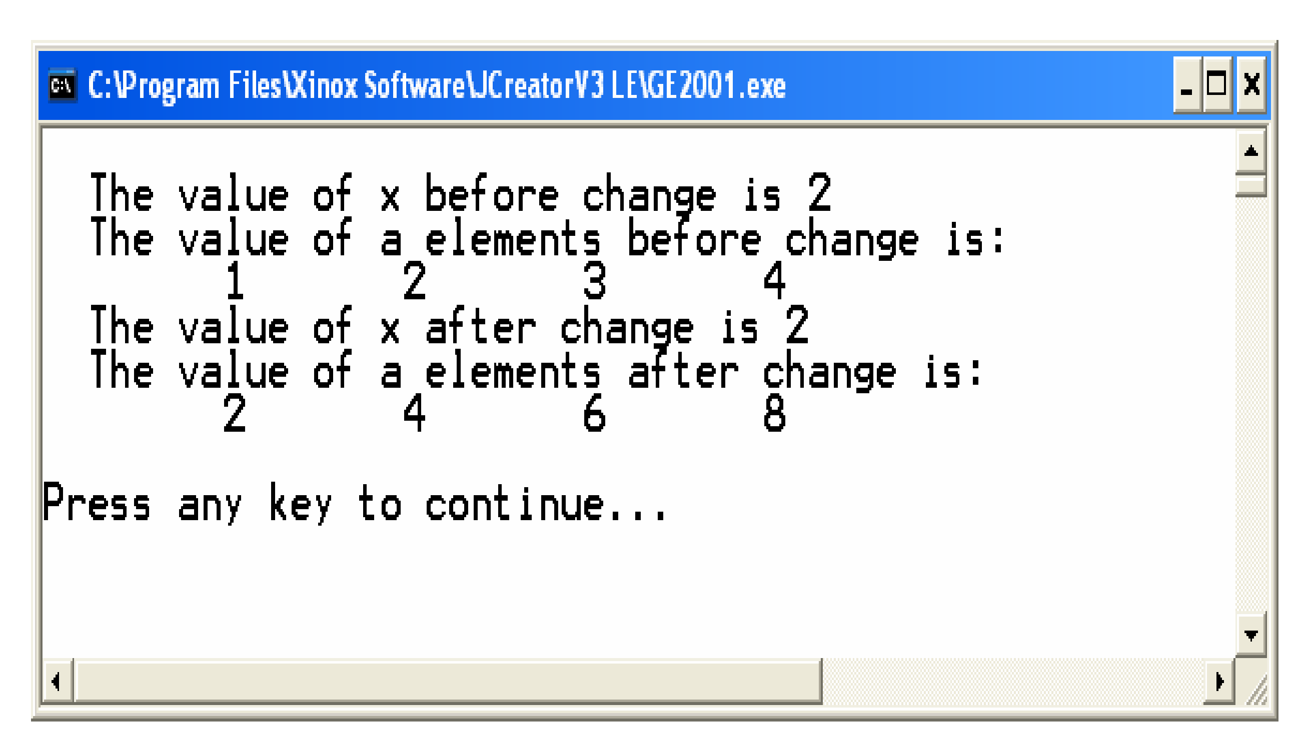
شرح المثال:

هذا المثال يبين كيفية الاستدعاء باستخدام القيمة والاستدعاء باستخدام العنوان. في السطر رقم (٩) تمت عملية استدعاء للطريقة change بإرسال x (متغير يشير إلى قيمة من نوع int) و a (متغير يشير إلى مصفوفة من نوع int). وكما ذكرنا سابقاً فإن عملية تمرير المتغيرات التي تشير إلى أنواع بيانات بدائية (Primitive Data Types) تكون باستخدام القيمة حيث سيتم إرسال نسخة من قيمة المعامل الفعلي x إلى المعامل الشكلي i الموجود في تعريف هذه الطريقة. وبما أنه قد تم إرسال نسخة من قيمة المعامل الفعلي إلى المعامل الشكلي فهذا يعني بأنهما يشيران إلى مكانين مختلفين في الذاكرة الرئيسية. أما بالنسبة للمعامل الشكلي الثاني a والذي يرسل عند استدعاء الطريقة change فهو عبارة عن مصفوفة. وكما نعرف بأن المصفوفة هي عبارة عن كائن (Objects) إذن سيتم إرسال عنوان هذه المصفوفة إلى المعامل الشكلي b، أي سيكون المعامل الفعلي a والمعامل الشكلي b يشيران إلى نفس الموقع (المصفوفة) في الذاكرة. والشكل (٢-١١) يوضّح الفرق بين عملية التمرير باستخدام القيمة وعملية التمرير باستخدام العنوان وذلك عند استدعاء الطريقة change الموجودة في السطر (٩) في المثال السابق.


١
قبل عملية الاستدعاء للطريقة Change
a
x
1
2
3
4
2
٢
عند عملية الاستدعاء للطريقة Change
a b
x
i
1
2
3
4
2
2
٣
خلال عملية التنفيذ والوصول لنهاية الطريقة Change
a b
x
i
2
4
6
8
2
4
٤
عند الخروج من الطريقة Change والعودة للطريقة main()
a
x
2
4
6
8
2

شكل (٢-١١)

والشكل (٢-١٢) يبين نتائج تنفيذ البرنامج في المثال (٢-٦).



شكل (٢-١٢)

الاستدعاء الذاتي (Recursion):

والمقصود بالاستدعاء الذاتي هو أن تقوم الطريقة باستدعاء نفسها نفسها بنفسها، فهنالك الكثير من المسائل التي يمكن حلها باستخدام الاستدعاء الذاتي وبهذه الحالة يمكن توفير الكثير من جمل التكرار فنستطيع الاستعاضة عن جمل التكرار بعملية استدعاء الطريقة لنفسها. فمثلا لإيجاد المضروب (factorial) لعدد معين يمكن كتابة البرنامج على الشكل التالي:

مثال (٢-٧):

```
// factorial.java

1.     import javax.swing.JOptionPane;
2.     public class factorial {
3.      public static void main (String args[ ]){
4.       String snum1;
```

```
5.      int num1, fac_of_num1;
6.      snum1 = JOptionPane.showInputDialog("Enter num1:") ;

7.     num1 = Integer.parseInt(snum1);
8.      fac_of_num1 = fact(num1);
9.     JOptionPane.showMessageDialog(null, num1 + "! = " +
                                                fac_of_num1);
10.    }      //end of main
11.
12.    static int fact(int n){
13.     if (n == 0 || n ==1)
14.       return 1;
15.     else
16.       return n * fact(n-1);
17.   }             //end of fact method
18.   }             //end of class factorial
```

شرح المثال:

في السطر (٦) يتم إدخال الرقم المراد حساب قيمة المضروب (Factorial) له. ثم في السطر (٨) يتم استدعاء الطريقة fact والتي تقوم بعملية حساب قيمة المضروب وتعيدها ليتم تخزينها في المتغير fac_of_num1. والآن لنرى ماذا سيحدث عند استدعاء الطريقة fact: في الاسطر (١٣-١٤) تتم عملية السؤال عن قيمة الرقم المرسل فإذا كان مساوياً للواحد أو للصفر (حسب التعريف الرياضي لعملية إيجاد المضروب) تتوقف هذه الطريقة وترجع واحد. ويعتبر هذا الشرط هو شرط التوقف الوحيد للاستدعاء الذاتي لهذه الطريقة، حيث أنه لابد من وجود شرط توقف داخل طرق الاستدعاء الذاتي وإلا استمرت عملية الاستدعاء الذاتي إلى مالانهاية. وفي السطر (١٦) تتم ارجاع العدد مضروباً بعملية استدعاء أخرى لنفس الطريقة ولكن هذه المرة بإرسال العدد السابق مطروحاً منه واحد (n-1)، وتستمر هذه العملية حتى يتحقق شرط الخروج من الطريقة وتوقيف عملية الاستدعاء الذاتي وتبدأ الآن عملية التعويض العكسي للقيم المرجعة من عمليات الاستدعاء. والشكل (٢-١٣) يوضّح عمليات الاستدعاء وعمليات التعويض العكسي عند إدخال الرقم ٤.

عمليات الاستدعاء الذاتى للطريقة fact

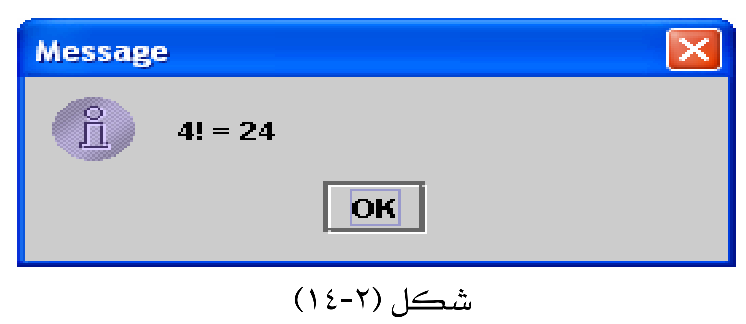
```
fact(3) * 4 = fact(4)
        ↓
fact(2) * 3
        ↓
fact(1) * 2
        ↓
        1
```

عمليات التعويض العكسى

```
        1
        ↓
fact(1) * 2
          1
        ↓
fact(2) * 3
      2 * 1
        ↓
fact(3) * 4
  3 * 2 * 1
        ↓
fact(4) = 24
```

شكل (٢-١٣)

والشكل (٢-١٤) التالي يبين مخرجات هذا البرنامج:



شكل (٢-١٤)

التحميل الزائد للطرق (Methods Overloading):

تتم عملية التحميل الزائد للطريقة عندما يكون هنالك أكثر من طريقة تحمل نفس الاسم في نفس الصنف ويتم التمييز بين هذه الطرق من خلال التوقيع (Signature) الخاص بكل منهم، أي إذا أردنا أن نقوم بتعريف أكثر من طريقة بنفس الاسم لابد أن تختلف هذه الطرق في: عدد المعاملات، نوع المعاملات، أو ترتيب المعاملات المختلفة الأنواع.

مثال (٢-٨):

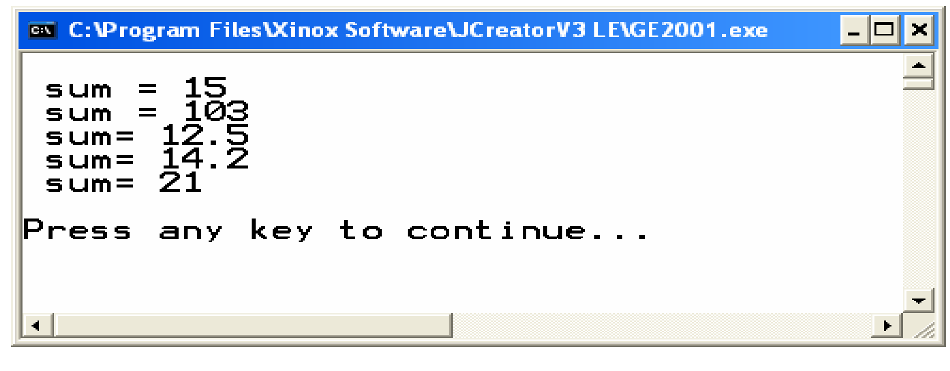
```
// Overload.java

1.    public class Overload {
2.    public static void main(String args[]){
3.    sum();
4.    sum(100,3);
5.    System.out.println("sum= " + sum(8.5, 4));
6.    System.out.println("sum= " + sum(10, 4.2));
7.    System.out.println("sum= " + sum(8, 9, 4));
8.    }
9.
10.   // no parametrs no return
11.   static  void sum () {
12.    int num1 = 10, num2 = 5;
13.    System.out.println("sum = " + (num1 + num2));
14.   }
15.   // has two parametes and no return
16.   static  void sum (int num1, int num2) {
17.    System.out.println("sum = " + (num1 + num2));
18.   }
19.
20.   // has two parametes   and return double
21.   static double sum( double num1, int num2) {
22.    return (double)(num1 + num2);
23.   }
24.   // has two parameters and return double
25.   // but different order of the parameters
26.   static double sum(int num2 ,double num1) {
```

```
27.    return (double)(num1 + num2);
28.   }
29.
30.   // has three parameters and return integer
31.   static int sum(int num1, int num2, int num3) {
32.    return num1+num2+num3;
33.   }
34.   }
```

شرح المثال:

يوضّح هذا المثال مفهوم التحميل الزائد للطرق (Methods Overloading). نلاحظ في هذا المثال وجود خمس طرق تحمل نفس الاسم sum، لكن لابد لهذه الطرق أن تختلف عن بعضها في واحد أو أكثر من مكونات التوقيع (وتحديداً يكون الاختلاف بنوع المعاملات، عدد المعاملات، أو ترتيب المعاملات المختلفة النوع). ففي السطر (١١) تم تعريف الطريقة sum حيث أنه ليس لها معاملات شكلية ولا ترجع قيمة من اي نوع (void). وفي السطر (١٦) تم تعريف طريقة ثانية لها نفس الاسم sum بحيث تحتوي على معاملين شكليين اثنين من نوع int ولا ترجع قيمة من أي نوع (void). بينما في السطر (٢١) تم تعريف طريقة ثالثة بنفس الاسم sum لكنها تحتوي على معاملين شكليين اثنين الاول من نوع double والثاني من نوع int ترجع هذه الطريقة قيمة من نوع double. وفي السطر (٢٦) تم تعريف طريقة رابعة لها نفس الاسم sum ولها معاملين شكليين الأول من نوع int والثاني من نوع double وترجع قيمة من نوع double. واخيراً في السطر (٣١) تم تعريف طريقة خامسة لها الاسم sum وتحتوي على ثلاث معاملات شكلية من نوع int وترجع قيمة من نوع int. والشكل (٢-١٥) يبين مخرجات هذا البرنامج بعد تنفيذه.



شكل (٢-١٥)

الطرق الخاصة بالسلاسل الرمزية (String):

السلاسل الرمزية(String) هي عبارة عن مجموعة من الرموز characters (حروف، ارقام، أو رموز خاصة) المتجاورة، وفي معظم لغات البرمجة تكون السلاسل الرمزية (String) على شكل مصفوفة من الرموز characters، ولكن في java تكون السلاسل الرمزية كـكائن منفصل من الصنف المسمى String ويحتوي هذا الصنف على العديد من الطرق الخاصة بمعالجة هذا النوع من البيانات ولكن هناك عملية واحدة يمكن استعمالها على السلاسل الرمزية (String) دون الحاجة إلى استدعاء أي طريقة ألا وهي عملية الدمج "+"، حيث تقوم هذه العملية بدمج سلسلتين رمزيتين لتكوين سلسلة رمزية واحدة. وتتم عملية تعريف المغيرات من نوع السلاسل الرمزية (String) وذلك باستخدام اسم الصنف String بدل نوع البيانات عند تعريف متغيرات من نوع بدائي (Primitive Data Types) كما في المثال التالي:

```
1. String s;
2. s= "Hello";
```

حيث تم في السطر (١) تعريف المتغير s من نوع الصنف String، وهنا يعتيبر s كـكائن من نوع الصنف String. وفي السطر (٢) تم تخزين قيمة ثابتة هي "Hello" من نوع السلاسل الرمزية في المتغير s. والجدول (٢-٢) يحتوي شرح لبعض الطرق (Methods) الخاصة بالسلاسل الرمزية.

إيجاد طول السلسلة الرمزية:	
`s.length()`	ترجع الطريقة **length()** طول السلسلة الرمزية **s**.
عمليات المقارنة بين سلستين رمزيتين (ملاحظة: لاتستخدم **==** و **=!**).	
`s.compareTo(t)`	تقوم الطريقة بمقارنة السلسلة الرمزية **s** مع السلسلة الرمزية **t** وتعيد رقم سالب اذا كانت **s** اقل من **t** وتعيد صفر إذا كانت **s** تساوي **t** وتعيد رقم موجب إذا كانت **s** أكبر من **t**.
`s.compareToIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة

	compareTo() ولكن مع اهمال حالة الحروف (صغيرة أم كبيرة).
s.equals(t)	تعيد true إذا كان s يساوي t.
s.equalsIgnoreCase(t)	تعمل هذه الطريقة بنفس عمل الطريقة equals() ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).
s.startsWith(t)	تعيد true إذا كان s يبدأ بالسلسلة الرمزية t.
s.startsWith(t, i)	تعيد true إذا كانت السلسلة الرمزية t موجودة في s بدءاً من الموقع i.
s.endsWith(t)	تعيد true إذا كان s تنتهي بِ t.

عمليات البحث:

كل طرق indexOf() تقوم بإرجاع 1- إذا كان العنصر المراد البحث عنه غير موجود، ويمكن للعنصر المراد البحث عنه أن يكون حرف أو سلسلة رمزية.

s.indexOf(t)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s.
s.indexOf(t, i)	ترجع موقع أول مكان توجد فيه t داخل السلسلة الرمزية s بعد الموقع i.
s.indexOf(c)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.
s.indexOf(c, i)	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s بعد الموقع i.
s.lastIndexOf(c)	ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير c داخل السلسلة الرمزية s.

`s.lastIndexOf(t)`	ترجع موقع آخر مكان توجد فيه السلسلة الرمزية **t** داخل السلسلة الرمزية **s**.
عمليات أخذ جزء من السلسلة الرمزية **string**.	
`s.charAt(i)`	ترجع الحرف الموجود في الموقع **i** داخل السلسلة الرمزية **s**.
`s.substring(i)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى النهاية.
`s.substring(i, j)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى الموقع **j-1**.
عمليات التعديل على السلسلة الرمزية **string** وإنشاء سلسلة رمزية جديدة.	
`s.toLowerCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف صغيرة.
`s.toUpperCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف كبيرة.
`s.trim()`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد الفارغ من البداية والنهاية.
`s.replace(c1, c2)`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد تبديل كل **c1** بـ **c2**، وهما من نوع **char**.
عمليات أخرى على السلاسل الرمزية **string**.	
`s.matches(regexStr)`	ترجع هذه الطريقة **true** إذا كانت السلسلة

	الرمزية **regexStr** تطابق السلسلة الرمزية **s** كاملة.
`s.replaceAll(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل كل **regexStr** بِـ **t**.
`s.replaceFirst(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل أول **regexStr** بِـ **t**.
`s.split(regexStr)`	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية **s** مقسمة حسب ظهور **regexStr**.
`s.split(regexStr, count)`	كما في الطريقة **split(regexStr)** لكن مع تحديد عدد مرات التقسيم.

جدول (٢-٢)

والمثال (٢-٩) التالي يوضّح تنفيذ معظم الطرق الموجودة في الجدول (٢-٢) السابق.

مثال (٢-٩):

```
// Strings.java
 1. public  class UseMath{
 2.  public static void main(String args[]){
 3.   String s0="Well Come to Java World!" ,
 4.   s = "hello", t = "HELLO", s1, s2[], s3;
 5.   char c;
 6.   boolean b;
 7.   int i;
 8.   System.out.println();
 9.   i = s0.length();
10.    System.out.println(" The length of " + "\"" + s0 + "\"" + " = " + I +
                                        "\n");
 11.   i = s.compareTo(t);
 12.   if (i == 0)
13.     System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
```

```
14.   else if (i<0)
15.   System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
16.   else
17.   System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
18.   i = s.compareToIgnoreCase(t);
19.   System.out.print(" Ignoring case: ");
20.   if (i == 0)
21.   System.out.println(" \"" + s + "\"" + " is == " + "\"" + t + "\"\n");
22.   else if (i<0)
23.   System.out.println(" \"" + s + "\"" + " is < " + "\"" + t + "\"\n");
24.   else
25.   System.out.println(" \"" + s + "\"" + " is > " + "\"" + t +"\"\n");
26.   b =  s.equals(t);
27.   System.out.println(" Is " + "\"" + s + "\"" + " equals to " + "\"" + t
                                    + "\"" + " ?  " + b +"\n");
28.   b = s.equalsIgnoreCase(t);
29.   System.out.print(" Is " + "\"" + s + "\"" + " equals to ");
30.   System.out.println("\"" + t + "\"" + " (ignoring case)? " + b +
                                    "\n");
31.   b = s.startsWith("H");
32.   System.out.println(" Is " + "\"" + s + "\"" + " starts with \"H\"? " +
                                    b + "\n");
33.   b = s.startsWith("l", 3);
34.   System.out.print(" Is " + "\"" + s + "\"" + " starts with \"l\" ");
35.   System.out.println("from position 3 ? " + b + "\n");
36.   b = s.endsWith("lo");
37.   System.out.print (" Is " + "\"" + s + "\"" + " ends with \"lo\"");
38.   System.out.println(" ,from position 3 ? " + b + "\n");
39.   i = s0.indexOf("Java");
40.   System.out.print (" Java is at position ");
41.   System.out.println( i + " of " + "\"" + s0 + "\"\n");
42.   i = s0.indexOf("java", 4);
43.   System.out.print(" java is at position" + i +"of ");
44.   System.out.println(" \"" + s0 + "\"" + ", starting from position
                                    4\n");
45.   i = s0.indexOf('e');
46.   System.out.print (" \'e\' is at position ");
```

```
47.   System.out.println(i + " of " + "\"" + s0 + "\"\n");
48.   i = s0.indexOf('e', 4);
49.   System.out.print (" \'e\' is at position " + i + " of  ");
50.   System.out.println("\"" + s0 + "\"" + " starting from position 4\n");
51.   i = s0.lastIndexOf('e');
52.   System.out.print(" Last occurrence  of \'e\' in ");
53.   System.out.println("\"" + s0 + "\"" + " is at "+ I + "\n");
54.   i = s0.lastIndexOf(t);
55.   System.out.print(" Last occurrence  of 'rl' in ");
56.   System.out.println("\"" + s0 + "\"" + " is at " + I + "\n");
57.   c = s0.charAt(3);
58.   System.out.print(" The character at position 3 in");
59.   System.out.println("\"" + s0 + "\"" + " is  " + c + "\n");
60.   s3 = s0.substring(6);
61.   System.out.print (" The substring of ");
62.   System.out.println("\"" + s0 + "\"" + " starting from 6 is\n" +
                                        "\t\t\t\"" + s3 + "\"\n");
63.   s1 = s0.substring(6, 10);
64.   System.out.print(" Substring of " + "\"" + s0 + "\"" + " starting ");
65.   System.out.println("from 6 to 10 is:" + "\"" + s1 + "\"\n");
66.   System.out.print (" \"" + s0 + "\"" + " in lowercase is ");
67.   System.out.println("\"" + s0.toLowerCase() + "\"\n");
68.   System.out.print("\""+ s0 + "\"" + "in uppercase ");
69.   System.out.println("\""+ s0.toUpperCase() + "\"\n");
70.   System.out.print(" \"" + s0 + "\"" + " with replacing all spaces ");
71.   System.out.println("with ';' is\n" + "\t\t\t\"" + s0.replace( '
                                        ',';')+"\"");
72.   System.out.println();
73.  }
74. }
```

والشكل (٢-١٦) يبين مخرجات هذا البرنامج الذي يوضّح بعض الطرق الخاصة بالسلاسل الرمزية والموجودة في الصنف (String).

```
The length of "Well Come to Java World!" = 24
"hello" is > "HELLO"
Ignoring case:  "hello" is == "HELLO"
Is "hello" equals to "HELLO" ?  false
Is "hello" equals to "HELLO" (ignoring case)? true
Is "hello" starts with "H"? false
Is "hello" starts with "l" from position 3 ? true
Is "hello" ends with "lo" ,from position 3 ? true
Java is at position 13 of "Well Come to Java World!"
java is at position -1 of  "Well Come to Java World!", starting from position 4
'e' is at position 1 of "Well Come to Java World!"
'e' is at position 8 of  "Well Come to Java World!" starting from position 4
Last occurrence  of 'e' in "Well Come to Java World!" is at 8
Last occurrence  of 'rl' in "Well Come to Java World!" is at -1
The character at position 3 in "Well Come to Java World!" is  l
The substring of "Well Come to Java World!" starting from 6 is
                        "ome to Java World!"
Substring of "Well Come to Java World!" starting from 6 to 10 is:"ome "
"Well Come to Java World!" in lowercase is "well come to java world!"
"Well Come to Java World!" in uppercase "WELL COME TO JAVA WORLD!"
"Well Come to Java World!" with replacing all spaces with ';' is
                        "Well;Come;to;Java;World!"
Press any key to continue...
```

شكل (٢-١٦)

تمارين:

س١: اكتب برنامجاً تطبيقياً بلغة جافا لإيجاد مساحة الدائرة:

$$Area = r^2 X \pi$$

(r : نصف القطر)

س٢: وضح الفرق بين المعاملات الشكلية (Formal parameters) والمعاملات الفعلية (Actual parameters or arguments).

س٣: وضح المقصود بما يلي:

أ - public method

ب -private method

س٤: ما هي الآلية التي من خلالها يتم إرجاع البيانات من الطريقة.

س٥: وضح باستخدام الأمثلة ما المقصود بما يلي:

أ - التمرير باستخدام القيمة Pass-By-Value

ب - التمرير باستخدام العنوان Pass-By-Reference

س٦: من المعلوم أن مجموع الأعداد من واحد إلى N يمثل بالمعادلة التالية:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + N - 1 + \sum_{i=1}^{N-2} i$$

اكتب برنامجاً تطبيقياً بلغة جافا لحل هذه المعادلة باستخدام:

أ - الاستدعاء الذاتي.

ب -جمل التكرار (الدوران).

س٧: اكتب برنامجاً تطبيقياً بلغة جافا لإيجاد حاصل ضرب iXj (ضرب الأعداد الصحيحة) حيث إن i>0، وذلك باستخدام عملية الجمع، مثلاً: 3X4 = 3+3+3+3 = 12.

س٩: اكتب طريقة تستقبل مصفوفة أعداد صحيحة وتعيد true إذا كانت جميع عناصر المصفوفة أعداد زوجية وتعيد false إذا كانت غير ذلك.

س١٠: وضح باستخدام مثال المقصود بالمصطلح Method Overloading.

س١١: اكتب برنامجا تطبيقيا بلغة جافا يحتوي على طريقتين (methods) الأولى تقوم بعملية تحويل درجات الحرارة المئوية Celsius إلى فهرنهايت Fahrenheit، حيث إن معادلة التحويل من المئوي إلى الفهرنهايتي هي:

$$F = 9.0 / 5.0 \;\; * \;\; (C + 32)$$

والطريقة الاخرى تقوم بعملية التحويل من الفهرنهايت Fahrenheit إلى المئوي Celsius، والمعادلة التالية تستخدم للتحويل من الفهرنهايتي إلى المئوي:

$$C = 5.0 / 9.0 \;\; * \;\; (F - 32)$$

س١٢: اكتب برنامجاً تطبيقياً بلغة جافا يحتوي على الطرق التالية (جميع الطرق تستقبل متغير من نوع (String):

أ -طريقة تعيد عدد الجمل في السلسلة الرمزية (يتم الفصل بين الجمل عند الادخال بالنقطة).

ب -طريقة تعيد عدد الكلمات في السلسلة الرمزية (يتم الفصل بين الكلمات بالفراغ).

ج -طريقة تعيد عدد الكلمات في كل جملة من الجمل التي تم معرفتها من خلال الطريقة في الفقرة (أ).

د -طريقة تعيد عدد الأحرف في كل جملة من الجمل التي تم معرفتها من خلال الطريقة في الفقرة (أ)، دون احتساب الفراغ أو علامات الترقيم.

هـ -طريقة تعيد متوسط عدد الكلمات لكل الجمل.

و -طريقة تعيد متوسط عدد الأحرف لكل الكلمات.

س١٣: اكتب ناتج تنفيذ البرامج التالية:

أ-

```
public class checkupper {
 public static void main (String args[]){
  char c1 = 'f', c2 = 'T';
  System.out.println("Is "+c1 +" in uppercase ? " +
                          isUpperCase(c1));
  System.out.println("Is "+c2 +" in uppercase ? " +
                          isUpperCase(c2));
 }
 static boolean isUpperCase(char testChar) {
  return ((testChar>='A') && (testChar<='Z'));
  }
}
```

ب –

```
public class validateAddress {
 public static void main (String args[]){
  String My_email = "java_doc@java.net";
  if (validate(My_email) == true)
   System.out.println("this a valid email address");
  else
   System.out.println("this an invalid email
                          address");
 }
 static boolean validate(String email) {
  String name;
  String domain;
```

```
  int index;
  if (( index = email.indexOf( '@')) == -1) {
   return false;
  }
  name = email. substring(0, index);
  domain=email.substring(index+1,email.length());
  System.out.println(" Name: " + name);
  System.out.println(" Domain: " + domain);
  return true;
 }
}
```

ج-

```
public class primenumbers{
 public static void main(String [] args) {
  System.out.println("The Prime numbers between 1 and
                   100 are");
  for (int i = 0; i < 100; i++)
   if (isPrime(i))
    System. out. print(i + " ");
 }
 static boolean isPrime(int test) {
  if (test < 2)
   return false;
  if (test == 2)
     return true;
  for (int i = 2; i < test; i++)
```

```
      if (( test % i) == 0)
        return false;
  return true;
 }
}
```

د-

```
public class SwapArray{
 public static void main(String [] args) {
  int values[]={1, 2, 3, 4, 5, 6, 7, 8};
  System.out.println("values before swap");
  printArray(values);
  swap(values);
  System.out.println ("values after swap");
  printArray(values);
 }
 static void swap(int a[]){
  int length = a.length, temp;
  for (int i = 0; i <= (length/2); i++){
   temp = a[ length - i - 1];
   a[length - i - 1] = a[ i];
   a[ i] = temp;
  }
 }
 static void printArray(int a[]){
  for (int i =0;i<a.length;i++){
   System.out.print (a[i]+"\t");
  }
```

```
  System.out.println( );
 }
}
```



برمجة ٢

الأصناف والكائنات

الأصناف والكائنات

٣

الجـــــدارة:

أن يكون المتدرب قادراً على التعامل مع الأصناف والكائنات في كتابة برامج لغة جافا.

الأهـــداف:

عندما تكمل هذه الوحدة تكون قادراً على:

١ - فهم ماهية البرمجة الكينونية (OOP) وكيفية الاستفادة منها.
٢ - تعريف الاصناف وتحديد مكوناتها من بيانات وطرق.
٣ - فهم واستخدام مفهوم الوراثة للأصناف وكيفية الوصول للطرق الموروثة.
٤ - استبدال الطرق الموروثة بطرق أخرى (Method Overriding).

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان هذه الجدارة بنسبة ١٠٠٪.

الوقت المتوقع للتدريب: ٨ ساعات.

الوسائل المساعـدة:

- قلم.
- دفتر.
- جهاز حاسب آلي.

متطلبات الجدارة:

اجتياز جميع الحقائب السابقة.

البرمجة الموجهة للكائنات (Object Oriented Programming):

تعتبر الأصناف (classes) والكائنات (objects) مفهومين أساسين في برمجة الكائنات وتكمن الفائدة في استخدام برمجة الكائنات في أن معظم برامج الحاسوب المستخدمة لحل المشاكل الحقيقية تكون كبيرة جدا وأكبر من تلك التي المستخدمة في الوحدات السابقة وثبت عمليا أن أفضل طريقة لكتابة البرنامج هي تقسيمه إلى وحدات صغيرة (modules) ويعرف هذا المبدأ بمبدأ "فرّق تسد" (divide and conquer).

وحدات البرنامج في لغة جافا هي الأصناف (classes). عندما يقوم الشخص بكتابة برنامج جديد يقوم بضم الأصناف (classes) الجديدة مع تلك المتوفرة في مكتبة الأصناف في جافا (API) ،وتتم عملية التخاطب بين هذه الأصناف عن طريق الرسائل (messages). حيث توفر هذه المكتبة العديد من الأصناف التي تقوم بالعمليات الحسابية ومعالجة النصوص وعمليات الإدخال والإخراج والكثير الكثير من العمليات المفيدة الأخرى.

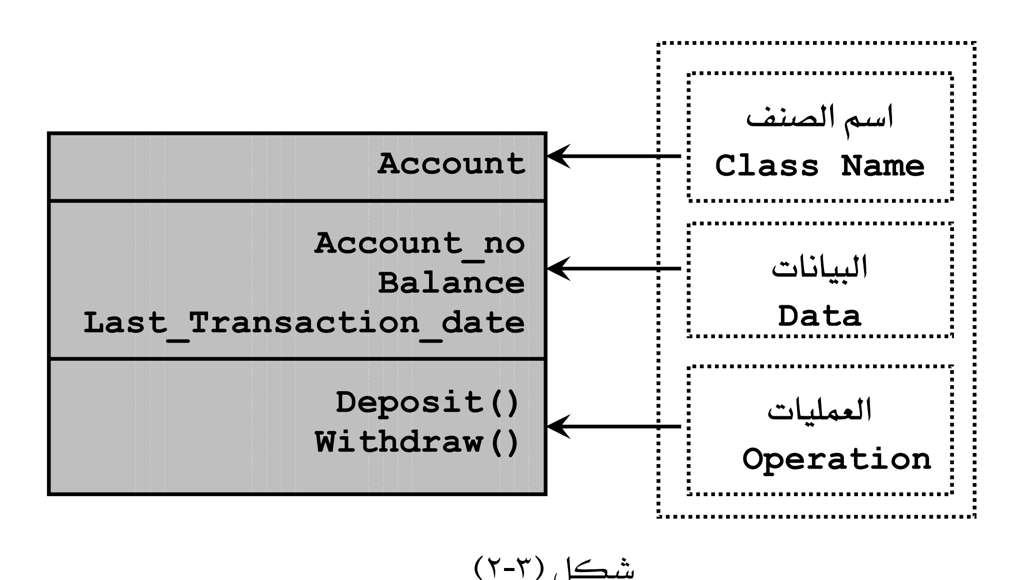
الكائن: هو عبارة عن شيء حقيقي في واقع الحياة العملية مثل الطالب محمد محمود احمد، الحساب رقم ١٢٢٣٢، السيارة التي تحمل اللوحة ك ا ن ١٠١، المريض خالد حسين . . . الخ، وكل هذه الأشياء تعتبر كائنات في بيئات مختلفة، فمثلا الطالب في البيئة الجامعية، الحساب في نظام البنك، السيارة في إدارة المرور، المريض في مستشفى، والى غير ذلك من الكائنات في بيئات العمل المختلفة.

الصنف: عبارة عن قالب (مخطط) يحتوي ويمثل الصفات للكائنات التي تنتمي لهذا الصنف، ويجب أن يحتوي هذا المخطط على جميع صفات الكائنات التي تنتمي إلية وجميع التفاصيل الخاصة بإنشاء هذه الكائنات (النسخ). .فمثلا الصنف "طالب" (Student) يمثل جميع الصفات لجميع الطلاب في بيئة معينة. والصنف "حساب" (Account) يمثل صفات جميع الحسابات في بنك معين. وهذه الصفات أو البيانات (Data) يتم تمثيلها في الأصناف بالمتغيرات بينما العمليات (Operation) يتم تمثييلها باستخدام الطرق (Methods). والشكل (٣-١) يبين العلاقة بين الصنف "منزل" وبين الكائنات الممكن أن تتبع له.



شكل (٣-١)

والشكل (٣-٢) يبين كيفية تمثيل الأصناف بشكل رسومي من خلال ما يسمى بـالـ Class Diagram، حيث يبين هذا الرسم شكل بسيط جداً من الصنف "الحساب البنكي"، والذي يحتوي على البيانات والعمليات.



شكل (٣-٢)

تعـــريف الصنـــــف (Class Declaration) وتحديـــد مكونـــاته:

يتم تعريف الأصناف في لغة جافا عن طريق استخدام الكلمة المحجوزة class، حيث يتبعها اسم الصنف، وعند اختيار اسم للصنف لابد من من تطبيق القواعد الخاصة بالأسماء (مثل: أسماء المتغيرات و أسماء الطرق) في لغة جافا. والمثال (٣-١) يبين كيفية تعريف الصنف Account، لكن دون وجود جمل تنفيذية لأنه للتوضيح فقط.

مثال (٣-١):

```
// Account.java
1.   import java.util.Date;
2.   public class Account {
3.     private int account_number;
4.     private double balance ;
5.     private Date last_transaction_date;

6.     public void deposit(double amt){
7.      //deposit code
8.     }
9.     public void withdraw(double amt){
10.     // withdraw code
11.    }
12.  }
```


شرح المثال:

كما نلاحظ في هذا المثال فإن عملية تعريف الأصناف تكون بالطريقة التالية:

-نبدأ باسم الصنف (class name) ويمكن أن يكون مسبوقا بكلمة public (وتعني عام) وهذا يعني أنه يمكن لأي صنف آخر أن يقوم بإنشاء نسخ (instances) من هذا الصنف، أما إذا لم نضع كلمة public في عملية التعريف فإن الأصناف داخل نفس الحزمة (Package) التي يوجد بها هذا الصنف هي وحدها تستطيع إنشاء نسخ (instances) من هذا الصنف .

-ثم بعد ذلك نبدأ بتعريف المتغيرات كما في الاسطر (٣-٥)، و كما نلاحظ فإن هذه المتغيرات مسبوقة بكلمة private (وتعني خاص) وهذا يعني أن هذه المتغيرات يمكن التعامل معا داخل هذا الصنف فقط، أما إذا كانت مسبوقة بكلمة Public فإن جميع الأصناف يمكنها التعامل مع هذه المتغيرات (بعد إنشاء نسخة instance من هذا الصنف) أما إذا لم نضع شيء فإن الأصناف داخل نفس الحزمة (Package) التي يوجد بها هذا الصنف هي وحدها تستطيع التعامل مع هذه المتغيرات (بعد إنشاء نسخة instance من هذا الصنف).

-وفي الأسطر (٧-٩) و الأسطر (١٠-١٢) تم تعريف العمليات (الطرق) على الصنف.

إنشاء الكائن (Object Creation) والوصول لمكوناته :

والآن بعد أن لاحظنا كيف يتم تعرف الأصناف لنرى كيف يتم استخدام هذه الأصناف: تتم عملية استخدام الأصناف وذلك عن طريق إنشاء كائنات (Objects) تكون على شكل نسخ من هذا الصنف) وبالتالي يتم التعامل مع هذه الكائنات (النسخ)، .وتتم عملية انشاء النسخ على النحو التالي:

-تعريف متغير من نوع الصنف المراد استخدامه والذي تم تعريفه مسبقاً.

-إنشاء كائن حقيقي من نفس الصنف وذلك باستخدام كلمة new متبوعة بإحدى البانيات (Constructors).

-ثم بعد ذلك يتم التعامل مع الكائن باستخدام اسم المتغير الذي يشير إليه متبوعا بنقطة ثم بأحد المتغيرات أو الطرق حسب امكانية الوصول (public, private, protected, default). حيث تم شرح طرق الوصول هذه في الوحدة الثانية، أما طريقة الوصول protected فهي تعني "محمي"، أي أن الطريقة أو متغير الصنف إذا عرف protected فهذا يعني أنه لا يمكن الوصول إليه إلا من خلال الصنف الذي عرفت فيه أو الاصناف المشتقة من هذا الصنف. والمثال (٣-٢) يبين كيفية انشاء كائن من الصنف Account الذي تم تعريفه في المثال (٣-١).

مثال (٣-٢)

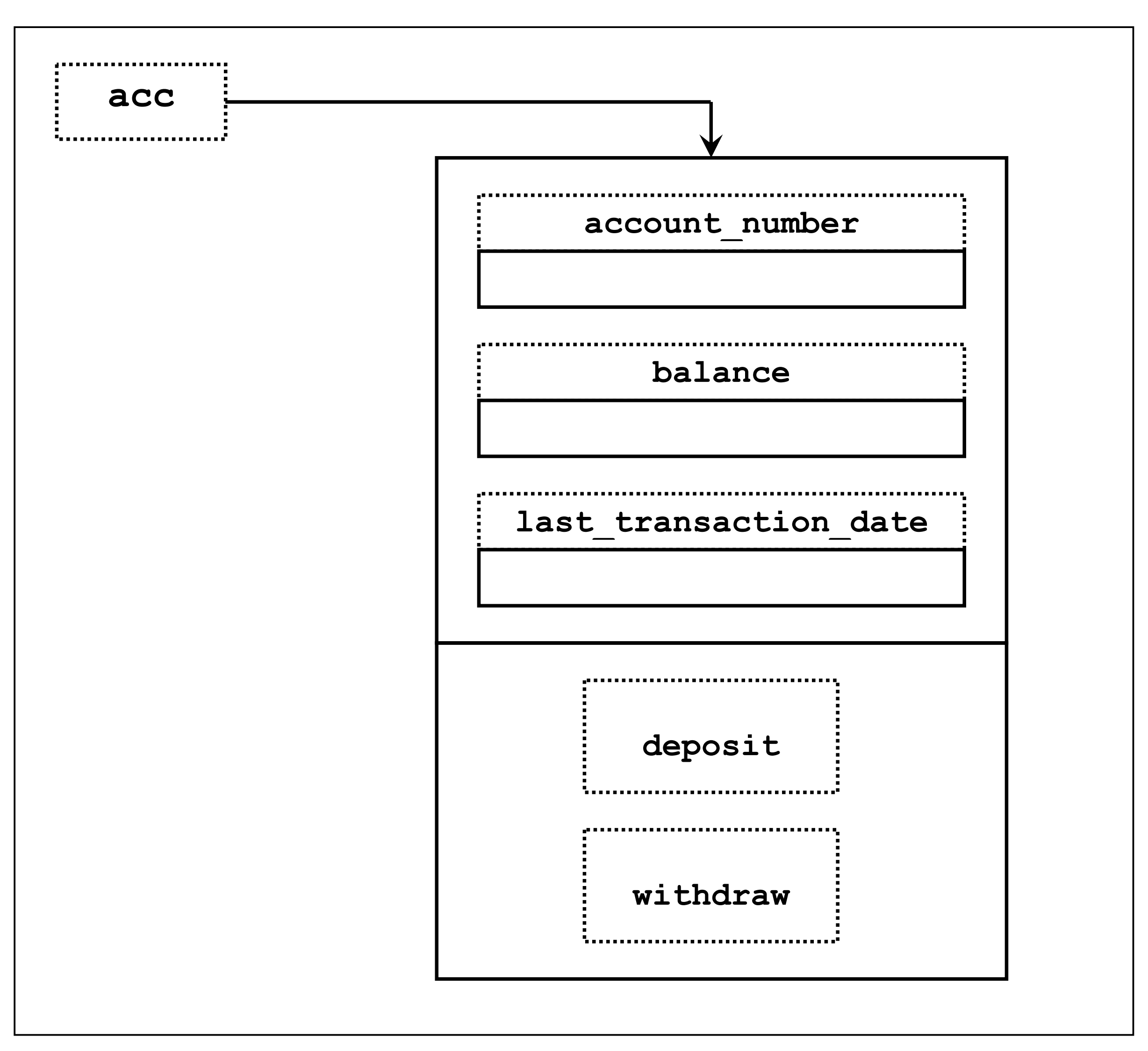
```
// Bank.java

1.  public class Bank {
2.   public static void main(String[] args) {
3.    Account acc =new Account();
4.    Acc.deposit(1000);
5.   }
6.  }
```

شرح المثال:

في السطر (٣) تم تعريف وإنشاء المتغير acc ليصبح كائن من نوع الصنف Account وهذا يعني أن المتغير acc يشير إلى كائن من نوع Account و الجملة new تقوم بإنشاء هذا الكائن بعد استدعاء احدى البانيات (Constructors) الخاصة بالصنف Account (والبانيات هي عبارة عن طرق تحمل نفس اسم الصنف بحيث يتم استدعائها عن إنشاء الكائن وتأتي بعد الكلمة المحجوزة new) وحجز الأماكن

اللازمة في الذاكرة لجميع المتغيرات. وفي السطر (٤) تمت عملية استدعاء الطريقة deposit في داخل الكائن acc، وذلك بكتابة اسم الكائن متبوعاً باسم الطريقة بحيث يفصل بينهما نقطة. والشكل (٣-٣) يبين محتويات الكائن acc.



شكل (٣-٣)

والبانيات (Constructors) كما ذكرنا سابقا هي عبارة عن طرق لها نفس اسم الصنف الذي عرّفت فيه، بحيث تستدعى عند إنشاء كائن من نوع هذا الصنف، وعندما تستدعى هذه البانيات فإنها تقوم بحجز مكان لهذا الكائن في الذاكرة وعادة ما تستخدم البانيات لإعطاء قيماً ابتدائية لمتغيرات الصنف، ويمكن للصنف الواحد أن يحتوي على أكثر من بانية وهذا ما يسمى بالتحميل الزائد للبانيات

(Overloaded Constructors)، وإذا لم نقم بتعريف بانية داخل الصنف فإنه يتم إنشاء البانية التلقائية (Default Constructor).

مثال (٣-٣):

```
// Account.java

1.     import java.util.Date;
2.     import javax.swing.JOptionPane;
3.      class Account{
4.       private int account_no;
5.       private String customer_name;
6.       private double balance;
7.       Date last_Transaction_Date;

8.      Account(int  no ,String name ){
9.       account_no=no;
10.      customer_name=name;
11.     }

12.     Account(int no ,String name ,double amt ){
13.      account_no=no;
14.      customer_name=name;
15.      balance=amt;
16.     }

17.     void deposit (double amt) {
18.      if (amt>0 ){
19.       balance += amt;
20.       last_Transaction_Date= new Date();
21.      }
22.      else
23.       JOptionPane.showMessageDialog(null,"the deposit amount  must
                                              be > 0");
24.     }

25.     void withdraw(double amt){
```

```
26.     if (amt<=balance ){
27.      balance-=amt;
28.      last_Transaction_Date= new Date();
29.     }
30.     else
31.      JOptionPane.showMessageDialog(null,"the withdraw amount
                                             must be <= balance");
32.    }

33.    public double getBalance(){
34.     return balance;
35.    }

36.    public String getCustomer(){
37.     return customer_name;
38.    }
39.   }
```

الملف الرئيسي القابل للتنفيذ حيث يحتوي على الصنف الذي بداخله الطريقة `main()`

```
// client_account.java
1.     public class client_account{
2.      public static void main(String args[]){
3.       Account acc1=new Account(12, "Ali");
4.       Account acc2=new Account(12, "Fahad", 7350.3);

5.       acc1.deposit(2341.5);
6.       acc2.withdraw(200);

7.       System.out.println("\n Name: "+acc1.getCustomer());
8.      System.out.println("\tHis Balance= " + acc1.getBalance());
9.      System.out.println("\tThe date of the last transaction is: " +
                                          acc1.last_Transaction_Date);

10.    System.out.println(" Name: "+acc2.getCustomer());
```

```
11.    System.out.println("\tHis Balance= " + acc2.getBalance());
12.    System.out.println("\tThe date of the last transaction is: " +
                                   acc2.last_Transaction_Date);
13.     System.out.println();
14.    }
15.  }
```

شرح المثال:

يتكون البرنامج في هذا المثال من صنفين منفصلين، الصنف الأول Account والمحفوظ في ملف اسمه Account.java، والصنف الثاني client_account والمحفوظ في الملف المسمى client_account.java وهو الصنف الرئيسي حيث يحتوي على الطريقة main() والمعرفة داخل الصنف client_account والذي تم فية تعريف كائنين من نوع الصنف Account هما acc1 و acc2.

في الصنف الأول Account.java في الأسطر (٤-٧) تم تعريف متغيرات الصنف، ثلاثة منها عرّفت بمحدد الوصول private والذي يمنع استخدام هذه المتغيرات الثلاثة بشكل مباشر خارج هذا الصنف. في الأسطر (٨-١١) والاسطر (١٢-١٦) تم تعريف بانيان بنفس الاسم لاكن يختلفات بعدد المعاملات الشكلية. حيث يمثّل هذا الصنف "حساب بنكي" ويحتوي على ما يلي:

-البيانات (Data):

١- رقم الحساب (account_no).
٢- اسم الشخص الذي يملك هذا الحساب (customer_name).
٣- رصيد الحساب (balance).
٤- تاريخ آخر عملية تمت على الحساب (last_Transaction_Date).

-البانيات (Constructors):

١ - باني لإنشاء حساب برقم الحساب واسم صاحب الحساب:
Account(int no, String name)

٢ - باني لإنشاء حساب برقم الحساب واسم صاحب الحساب ورصيد ابتدائي:
Account(int no, String name, double amt)

-العمليات (Methods):

١ - عملية السحب، يجب أن يكون المبلغ المسحوب أقل أو يساوي الرصيد الحالي (deposit).

٢ - عملية الإيداع، يجب أن يكون المبلغ المودع أكبر من صفر (withdraw).

٣ -معرفة الرصيد الحالي (getBalance).

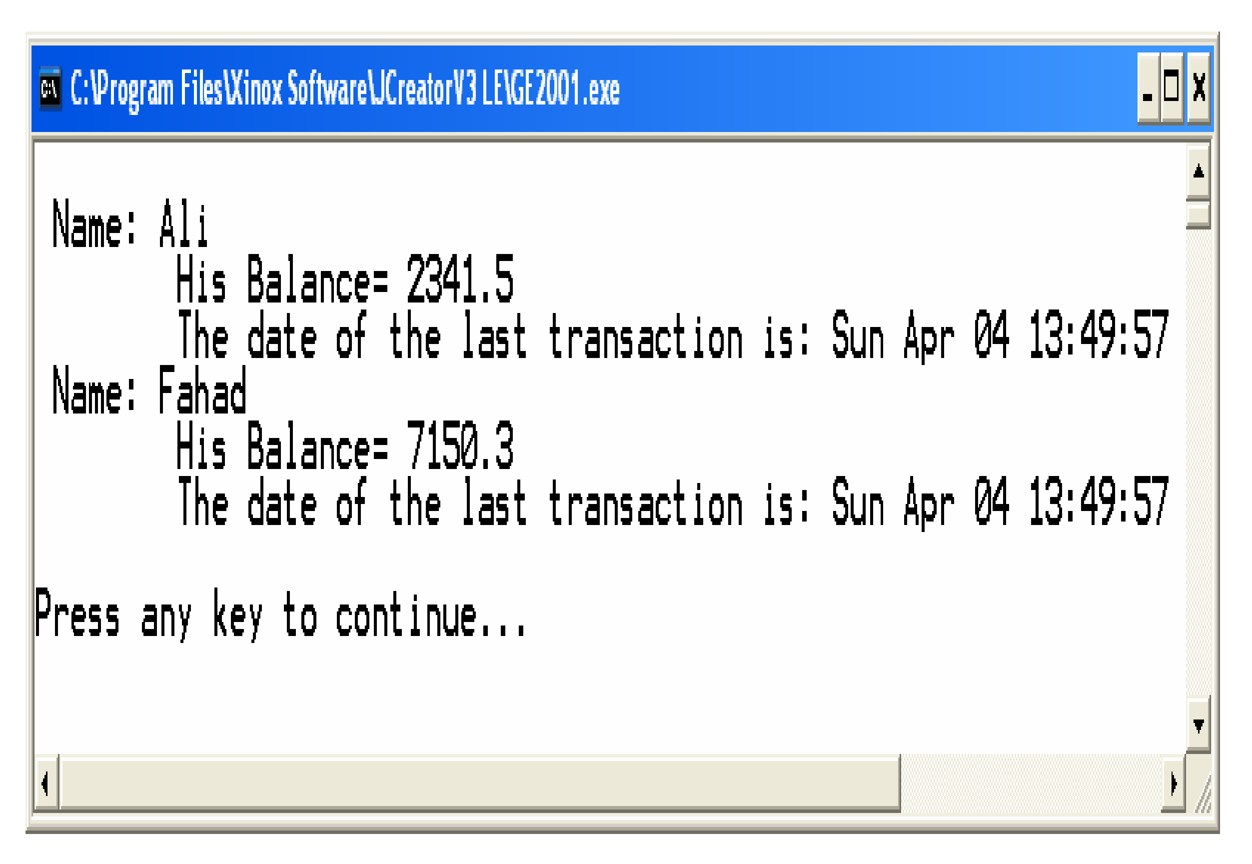
٤ -استرجاع اسم صاحب الحساب (getCustomer).

والشكل (٣-٤) يبين الشكل الرسومي للصنف Account.

```
Account
---------------------------------
int account_no
String customer_name
double balance
Date last_Transaction_Date
---------------------------------
Account(int,String)
Account(int,String,double)
void deposit (double)
void withdraw(double)
double getBalance()
String getCustomer()
```

شكل (٣-٤)

في الصنف الثاني (الرئيسي) client_account في الاسطري (٣-٤) تم إنشاء كائنين من نوع الصنف Account وهما acc1 و acc2، حيث تم استدعاء الباني الذي يحتاج إلى معاملين فعليين للكائن acc1 والباني الذي يحتاج إلى ثلاث معاملات فعليه للكائن acc2. وفي الاسطري (٥-٦) تم استدعاء طرق تابعة لكل من الكائنين. ومن خلال الاسطر (٩ و ١١) تم الوصول لمحتويات متغيرات الصنف المسموح الوصول إليها. والشكل (٣-٥) يبين مخرجات البرنامج في المثال السابق.



شكل(٣-٥)

وليتمكن المبرمج من إعادة استخدام الأصناف التي كتبها سابقاً دون الحاجة إلى إعادة كتابتها من جديد، لابد من وضع هذه الأصناف في حزمة (Package)، والحزمة (Package) هي عبارة عن حاوية (container) تحتوي على مجموعة من الأصناف المترابطة مع بعضها البعض ترابطا منطقيا. ومن فوائد استخدام الحزم أيضا إمكانية استخدام نفس الاسم لأكثر من صنف حيث أنه يمكن أن يكون لدينا الكثير من الأصناف التي تحمل نفس الاسم فيمكن أن نضع هذه الأصناف في حزم مختلفة وبالتالي يمكن استخدام أكثر من صنف يحمل نفس الاسم في مكان واحد.

وتتم عملية إنشاء الأصناف داخل حزمة وذلك بوضع الكلمة المحجوزة package في بداية الملف الذي يحتوي تعريف الصنف او الأصناف متبوعة باسم الحزمة، وبعد عملية الترجمة الناجحة للبرنامج يتم تخزين الأصناف وتحديداً الملفات ذات الإمتداد class في هذه الحزمة. وإذا لم تكن هذه الحزمة موجودة يتم إنشاؤها بعد انتهاء عملية الترجمة. والمثال (٣-٤) يوضّح عملية إنشاء الحزم (Packages).

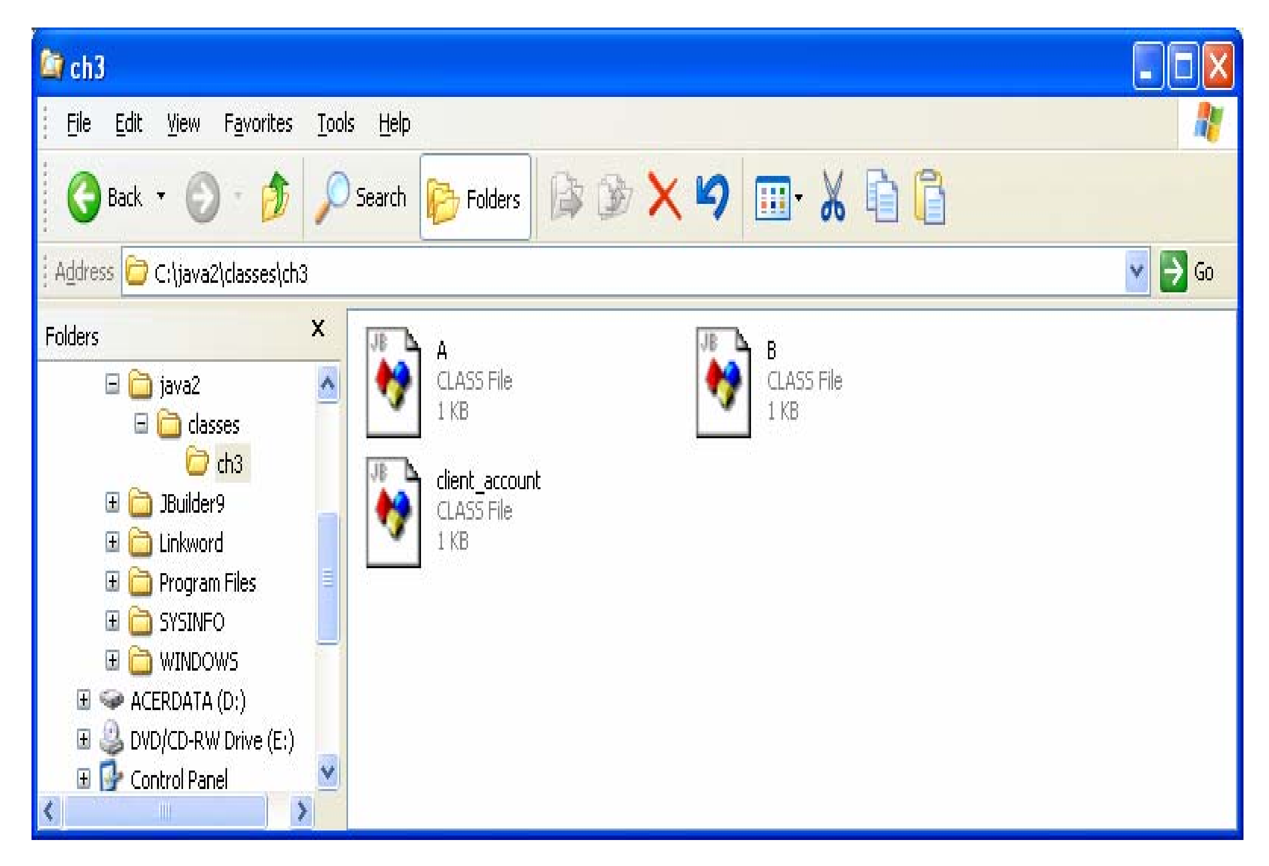
مثال (٣-٤):

```
// client_account.java

1.    package java2.classes.ch3;
2.
3.    class A{
4.      .
5.      .
6.      .
7.    }
8.
9.    class B{
10.     .
11.           .
12.     .
13.   }
14.
15.    public class client_account{
16.           public static void main(String args[]){
17.            .
18.      .
19.      .
20.           }
21.   }
```

شرح المثال:

في السطر (١) تم تحديد اسم ومسار الحزمة (Packages) التي ستحوي على جميع الأصناف التي سيتم إنشائها بعد ترجمة هذا المثال. بحيث سيتم تخزين الأصناف: A.class، B.class، و client_account.class في المجلد ch3 الموجود داخل المجلد classes والموجود داخل المجلد java2. لاحظ في هذا المثال وجود أكثر من صنف في نفس الملف وبعد الترجمة سيتحول كل صنف إلى ملف منفصل ذو امتداد class، لكن يجب أن لا يحتوي الملف الواحد على أكثر من صنف معرف كصنف عام public. والشكل (٣-٦) يبين الأصناف والمجلدات بعد ترجمة هذا المثال.



شكل (٣-٦)

مفهوم الوراثة (Inheritance):

تتم عملية الوراثة بين الأصناف من خلال اشتقاق صنف من صنف آخر، ففي هذه الحالة يرث الصنف المشتق (SubClass) جميع محتويات الصنف المشتق منه (Super Class) باستثناء المحتويات المعرّفة على أنها خاصة (private). وتتم عملية اشتقاق الأصناف من بعضها باستخدام الكلمة المحجوزة extends. والمثال (٣-٥) يبين عملية اشتقاق صنف من صنف آخر. ومن الجدير بالذكر أن لغة جافا لا تدعم الوراثة المتعددة، أي أن الصنف لا يمكن أن يرث أكثر من صنف واحد فقط.

مثال (٣-٥):

```
// y.java
1.    class x{
2.     .
3.     .
4.     .
5.    } // end of class x
6.    public class y extends x{
7.     .
8.     .
9.     .
10.  }
```

شرح المثال:

في هذا المثال تم تعريف الصنف x ومن ثم تم تعريف الصنف y المشتق من الصنف x والذي سيرث كل محتوياته، ونستطيع أن نقول أن الصنف x هو الأب والصنف y هو الابن، حيث سيرث الابن بعض أو كل صفات الأب. وفي هذا المثال التوضيحي لم نعرّف محتويات أي من الصنفين.

الوصول للطرق والبيانات الموروثة:

يتم الوصول للطرق والبيانات (المخزنة في متغيرات الصنف) بشكل مباشر داخل الصنف المشتق وكأنها معرفّة داخلة. ويجب أن نتذكر باننا لا نستطيع الوصول للطرق والمتغيرات المعرفة على أنها خاصة (private) بالصنف الذي عرّفت فيه، حيث أنها لا تورث للأصناف المشتقة من هذا الصنف. والمثال (٢-٦) يبين كيفية الوصول للطرق والبيانات (المتغيرات) الموروثة.

مثال (٣-٦):

```
// Cars.java

1.    class Transportation{
2.     protected static int x=12;
3.     private int y=19;
4.     public static void meth1(){
5.      System.out.println("Calling meth1() from class Cars.");
6.     }

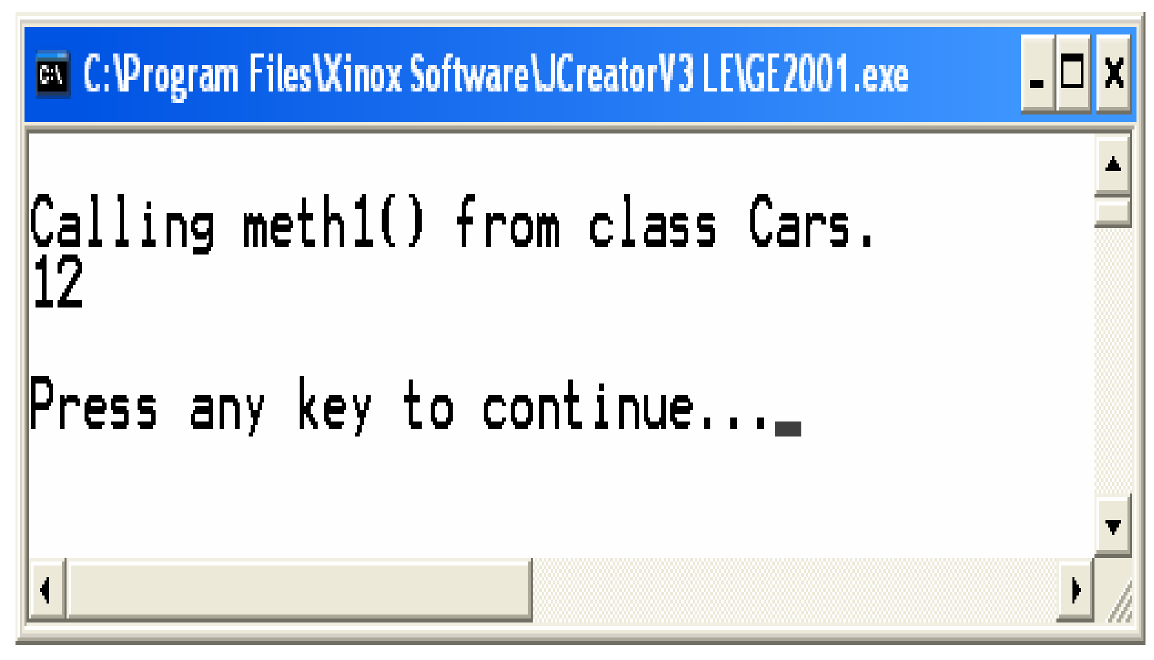
7.     private void meth2(){
8.      System.out.println("will not be called from Cars");
9.     }
10.   } // end of class Transportation

11.   public class Cars extends Transportation{
12.    public static void main(String args[]){
13.     meth1();
14.     System.out.println(x);
15.     //     meth2(); // meth2() has private access in
               // Transportation
16.     // System.out.println(y); // y has private access
                        // on Transportation
17.    }
18.   } // end of class Cars
```

شرح المثال:

في الأسطر (١-١٠) تم تعريف الصنف Transportation ليحتوي على متغيرين صنف هما: المتغير x وهو متغير محمي (protected) والمتغير y وهو متغير خاص (private)، ويحتوي هذا الصنف أيضاً على طريقتين هما: الطريقة meth1() وهي طريقة عامة (public) والطريقة meth2() وهي طريقة خاصة (private). وفي الاسطر (١٢-١٩) تم تعريف صنف ثاني هو الصنف Cars، بحيث تم اشتقاق هذا الصنف Cars من الصنف Transportation، وفي هذه الحالة يعتبر الصنف Transportation هو الـــ

Super Class و الصنف Cars هو الـــ SubClass، ومن منظور آخر نستطيع القول بأن الصنف Transportation هو الأب والصنف Cars هو الأبن. وفي هذا المثال يتم توريث جميع صفات (وهذه الصفات هي متغيرات الصنف و الطرق) الاب Transportation إلى الابن Cars باستثناء الصفات المعرفة على أنها خاصة private. وفي الاسطري (١٤-١٥) تم استخدام المتغير x والطريقة meth1() والمعرفين في صنف الأب (Transportation) وذلك لأنه تم توريثهم لصنف الابن (Cars)، بينما في الاسطري (١٦-١٧) لم نستطع استخدام المتغير y والطريقة meth2() لانهما معرفان بكونهما خاصين (private) بالصنف الذي تم تعريفهم فيه (Transportation)، وعند محاولة استخدامهم يظهر خطأ كما هو موضّح في المثال السابق. والشكل (٣-٧) يبين مخرجات هذا المثال.



شكل (٣-٧)

وعندما تقوم الطريقة method المعرفة داخل الصنف باستخدام احد مكونات هذا الصنف ففي بعض الأحيان لا يوجد هناك ما يثبت أن هذه العملية تجري على المكون الصحيح، لذلك فإن لغة جافا توفر المؤشر this الذي يشير إلى النسخة الحالية من الصنف وبالتالي فإن استخدام this يضمن أن تتم العملية على المكون الصحيح . والمثال (٣-٧) يوضّح كيف يتصرف البرنامج بدون استخدام المؤشر this، بينما يوضّح المثال (٣-٨) الفائدة من استخدام المؤشر this.

مثال (٣-٧):

```
// C.java

1.   class A{
2.     protected int a=9;
3.   } // end of class A

4.    class B extends A{
5.     void test(){
6.             int a=22;
7.             System.out.println("a = "+a);
8.     }
9.   } // end of class B

10.  public class C{
11.    public static void main(String args[]){
12.            B acc=new B();
13.            acc.test();
14.    }
15.  } // end of class C
```

شرح المثال:

تكون مخرجات هذا المثال هي طباعة ما يلي: a=22، وذلك لأن الجملة في السطر (٧) تتعامل مع المتغير a التابع للطريقة test() وليس المتغير a التابع للصنف A.

مثال (٣-٨):

```
// C.java

1.   class A{
2.     protected int a=9;
3.   } // end of class A

4.   class B extends A{
5.     void test(){
```

```
6.          int a=22;
7.          System.out.println("a = "+this.a);
8.     }
9.  } // end of class B

10. public class C{
11.    public static void main(String args[]){
12.          B acc=new B();
13.          acc.test();
14.    }
15. } // end of class C
```

شرح المثال:

بينما تكون مخرجات هذا المثال هي طباعة ما يلي: a=9، وذلك لأن الجملة في السطر (٧) تتعامل مع المتغير a التابع للصنف A وليس المتغير a التابع للطريقة test()، وذلك باستخدام المؤشر this والذي حدد أن المتغير المقصود هو المتغير a الموجود في الصنف الحالي B وبما أن الصنف B لا يحتوي على متغير باسم a فلذلك يتم استخدام المتغير a التابع للصنف A الذي تم اشتقاق الصنف الحالي منه.

بينما إذا احتوى الصنف المشتق (SubClass) والصنف المشتق منه (Super Class) على متغير أو طريقة بنفس الاسم، ففي هذه الحالة سواء تم استخدام المؤشر this أو لم يتم أستخدامة داخل الصنف المشتق (SubClass) فإنه يتم الرجوع للطريقة أو المتغير التابع لهذا الصنف المشتق (SubClass) ولن يتم الرجوع بأي حال من الأحوال إلى الطريقة أو المتغير المعرّف داخل الصنف المشتق منه (Super Class). وعند الحاجة للرجوع من داخل الصنف المشتق (SubClass) للطريقة أو المتغير التابع للصنف المشتق منه (Super Class) مع وجود تعريف لطريقة أو لمتغير بنفس الاسم داخل الصنف الحالي (SubClass) لابد من استخدام المؤشر super والذي يمكّنا من استخدام محتويات الصنف المشتق منه (Super Class). والمثال (٣-٩) يوضّح الفرق بين استخدام وعدم استخدام المؤشر super، حيث إن نتائج التنفيذ تختلف في كل حالة من الحالات.

مثال (٣-٩):

```
// C.java

1.    class A{
2.          void test1(){
3.     System.out.println("The test1() method was invoked FROM class
                              A");
4.          }
5.    } // end of class A

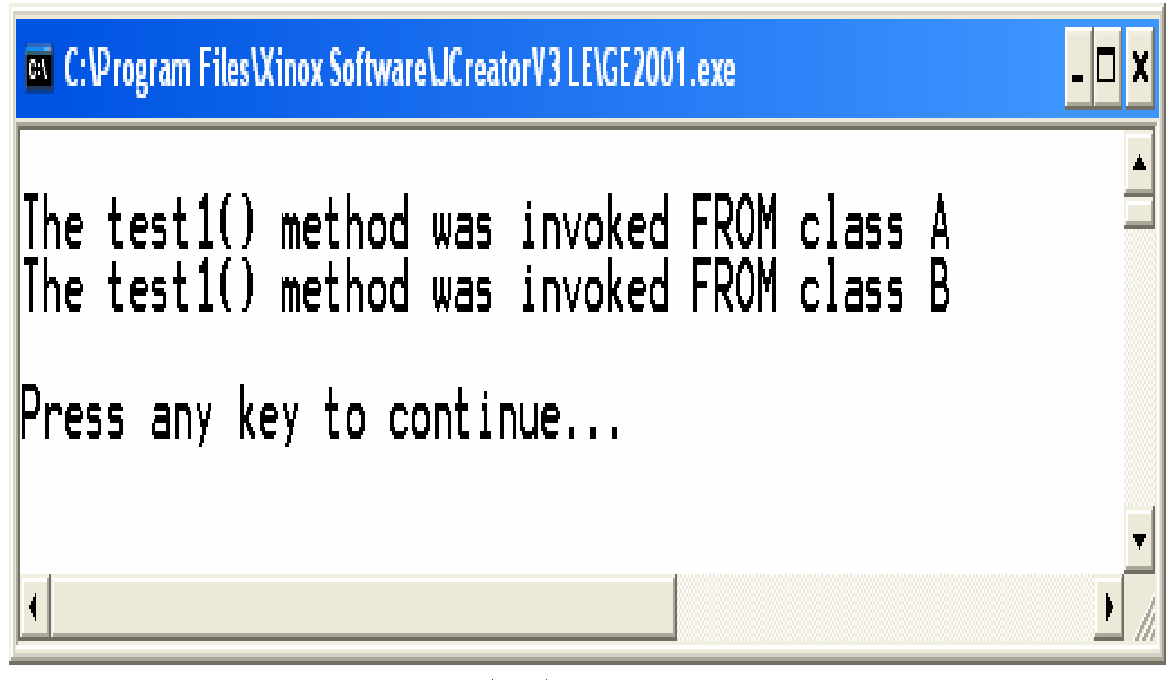
6.    class B extends A{
7.          void test(){
8.           super.test1();
9.           test1();
10.         }
11.         void test1(){
12.    System.out.println("The test1() method was invoked FROM class
                              B");
13.         }
14.   } // end of class B

15.   public class C{
16.         public static void main(String args[]){
17.          B acc=new B();
18.          acc.test();
19.         }
20.   } // end of class C
```

شرح المثال:

في السطر (٩) الموجود في الصنف B المشتق من الصنف A، تم استدعاء الطريقة ()test1 والمعرّفة في كل من الصنف B والصنف A. وفي هذا السطر تم تحديد استدعاء الطريقة ()test1 والمعرّفة في الصنف المشتق منه A، وتم هذا التحديد عن طريق استخدام المؤشر Super. بينما في السطر (١٠) تم استدعاء الطريقة ()test1 دون استخدام المؤشر Super وبالتالي سيتم تنفيذ الطريقة ()test1 المعرفة داخل الصنف

الحالي (الصنف الذي تمت عملية الاستدعاء منه) B. والشكل (٣-٨) يبين المخرجات عند تنفيذ البرنامج المكتوب في هذا المثال.



شكل(٣-٨)

استبدال الطرق الموروثة (Methods Overriding):

قد نحتاج في بعض البرامج إلى استبدال وتغيير طبيعة عمل الطرق الموروثة من صنف الاب (Super Class) إلى صنف الابن (SubClass)، ففي هذه الحالة لابد لنا من استبدال الطريقة الموروثة (Methods Overriding) وذلك بإعادة تعريف هذه الطريقة بالشكل الذي نريد، وإذا احتجنا إلى الوصول للطريقة الموروثة لابد من استخدام المؤشر super كما لاحظنا سابقاً من هذه الوحدة.

مثال (٣-١٠):

```
// Test.java

1.      class Car {
2.       private int year;
3.       private float originalPrice;
```

```
4.      // calculate the sale price of a car based on its
      // cost
5.      public double CalculateSalePrice() {
6.       double salePrice;
7.       if (year > 1994)
8.        salePrice = originalPrice * 0.75;
9.       else if (year > 1990)
10.       salePrice = originalPrice * 0.50;
11.      else
12.       salePrice = originalPrice * 0.25;
13.      return salePrice;
14.     }

15.     // a public constructor
16.     public Car(int year, float originalPrice) {
17.      this.year = year;
18.      this.originalPrice = originalPrice;
19.     }
20.    }

21.    class ClassicCar extends Car {
22.     // calculate the sale price of a car based on its
      // cost
23.    public double CalculateSalePrice() {
24.      return 10000;
25.     }

26.     // a public constructor
27.     public ClassicCar(int year, float originalPrice) {
28.      super(year, originalPrice);
29.     }
30.    }

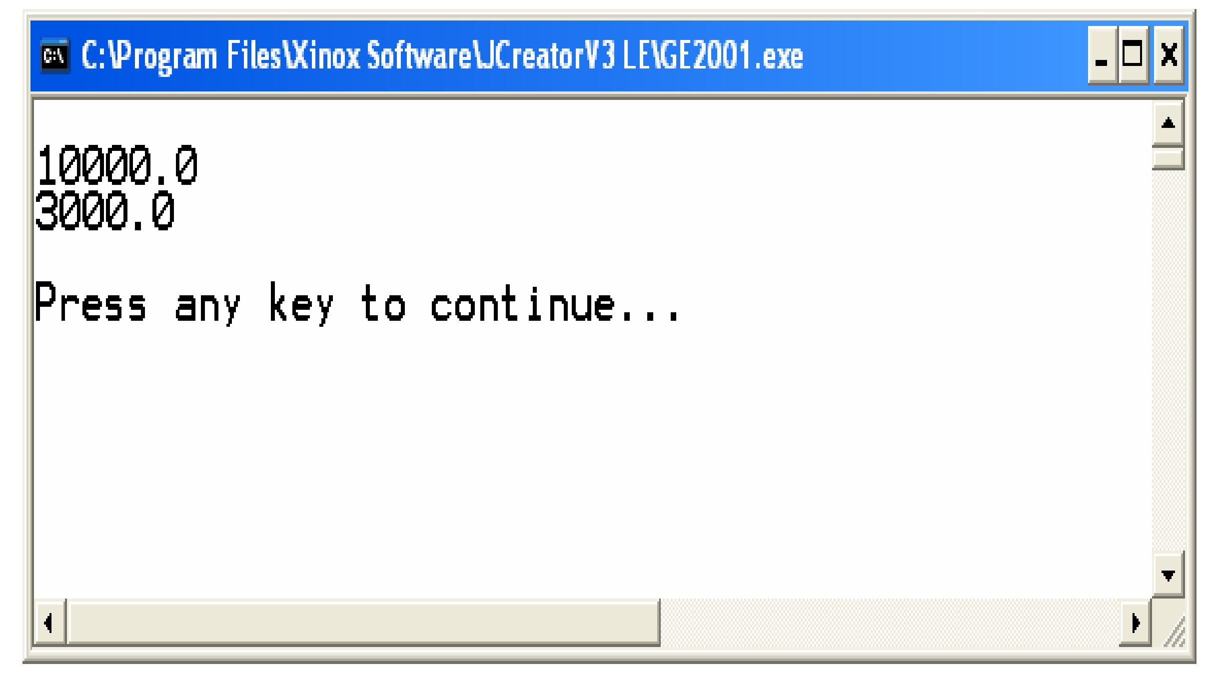
31.    public class Test{
32.     public static void main(String args[]){
33.      ClassicCar myClassic = new ClassicCar(1920, 1400);
34.      double classicPrice =
```

```
              myClassic.CalculateSalePrice();
35.     System.out.println(classicPrice);

36.     Car myCar = new Car(1990, 12000);
37.     double price = myCar.CalculateSalePrice();
38.     System.out.println(price);
39.   }
40.  }
```

شرح المثال:

في ألاسطر (٥-١٤) تم تعريف الطريقة CalculateSalePrice() بالنسبة للصنف Car، وفي الصنف ClassicCar المشتق من الصنف Car احتجنا إلى تغيير عمل الدالة الموروثة CalculateSalePrice() مما دعانا إلى إعادة تعريفها بالنسبة للصنف ClassicCar، وتمت عملية إعادة التعريف في الأسطر (٢٣-٢٥) وهذا مايسمى باستبدال الطرق الموروثة (Methods Overriding). لاحظ في السطر (١٧) اضطررنا إلى استخدام المؤشر this وذلك للتميز بين المعامل الشكلي year والمتغير year الموروث من الصنف Car. والشكل (٣-٩) يبين مخرجات البرنامج في هذا المثال.



شكل (٣-٩)

تمارين:

س١: أنشئ صنف وسمه Rational لتنفيذ العمليات الحسابية على الكسور، بحيث يحتوي هذا الصنف على ما يلي:

-متغيرات الصنف: متغيرين خاصين (private) من نوع int هما: numerator (ليحتوي على البسط) و denominator (ليحتوي على المقام).

-باني (Constructor) يسمح بإعطاء قيمة ابتدائية (يجب أن يتم تخزين القمية الابتدائية للكسر بشكل مختصر، مثلاً: يتم تخزين 1/2 بدلاً من 2/4) للكسر عند تعريف كائن (Object) من نوع هذا الصنف.

-طريقة لجمع رقمين كسريين وتخزين الناتج بشكل مختصر.

- طريقة لطرح رقمين كسريين وتخزين الناتج بشكل مختصر.

-طريقة لضرب رقمين كسريين وتخزين الناتج بشكل مختصر.

-طريقة لقسمة رقمين كسريين وتخزين الناتج بشكل مختصر.

- طريقة لطباعة الرقم الكسري بالشكل التالي: a/b حيث a تمثل البسط (numerator) و b تمثل المقام (denominator).

- طريقة لطباعة الرقم الكسري على شكل رقم حقيقي (float)، وذلك بقسمة البسط على المقام.

ثم اكتب برنامج يستخدم هذا الصنف.

س٢: أنشئ صنف يمثل مربع وسمه Rectangle، بحيث يحتوي على المتغيرات التالية: length و width يأخذ كل واحد منهم قيمة ابتدائية تساوي ١، ويحتوي على طريقتين، تقوم الطريقة الأولى بحساب مساحة المربع، بينما تقوم الطريقة الثانية بحساب محيط المربع. ثم اكتب برنامج لتطبيق هذا الصنف.

س٣: أنشئ صنف مسمى HugeInteger لتمثيل عدد صحيح كبير جداً وذلك باستخدام مصفوفة مكونه من أربعين موقع كل موقع يحتوي على خانة واحدة من خانات هذا الرقم. ويحتوي هذا الصنف على الطرق التالية، (اكتب برنامج لتطبيق هذا الصنف):

-الطريقة inputHugeInteger لإدخال الرقم الصحيح الكبير جداً إلى المصفوفة.

- الطريقة outputHugeIntege لطباعة الرقم الصحيح الكبير جداً على الشاشة.
- الطريقة addHugeIntege لجمع عددين صحيحين كبيرين جداً.
- الطريقة subtractHugeIntege لطرح عددين صحيحين كبيرين جداً.
-الطريقة isEqualTo للسؤال فيما إذا كانا عددين صحيحين كبيرين جداً متساويين أم لا ، بحيث ترجع true إذا كانا متساويين وترجع false إذا كانا غير متساويين.
- الطريقة isGreaterThan للمقرنة بين العددين الصحيحين وتحيد هل الاول أكبر من الثاني أم لا، بحيث ترجع true إذا كان العدد الصحيح الاول أكبر من العدد الصحيح الثاني وترجع false إذا كان غير ذلك.

س٤ : أنشئ صنف لتمثيل التاريخ بحيث يسمى Date وله الخصائص التالية :

-يتم إخراج التاريخ بأحد الأشكال التالية :

MM/DD/YYYY
April 01, 2004
DDD YYYY

-استخدم التحميل الزائد للبانيات لإنشاء كائنات تحمل قيماً ابتدائية للتاريخ حسب أشكال التاريخ السابقة.

ثم اكتب برنامج يستخدم كائنات من نوع هذا الصنف.

س٥ : أنشئ صنف يسمى IntegerSet ، بحيث يحتوي كل كائن من هذا الصنف على أرقام صحيحة بين الصفر و ١٠٠، ويتم تمثيل مجموعة الأعداد الصحيحة داخلياً كمصفوفة من نوع boolean، وعندما تكون قيمة عنصر المصفوفة a[i] مساوية لِـ true يكون الرقم i من عناصر المجموعة، بينما إذا كانت قيمة العنصر a[j] تساوي false يكون الرقم j غير موجود في هذه المجموعة. ويقوم الباني الذي ليس له عوامل شكلية بإنشاء مجموعة فارغة، أي أن جميع عناصر المصفوفة تحتوي على القيمة false. كما ويحتوي هذا الصنف على الطرق التالية :

-الطريقة unionOfIntegerSet تنشئ مجموعة ثالثة تحتوي على ناتج تنفيذ عملية الاتحاد بين مجموعتين.

- الطريقة intersectionOfIntegerSet تنشئ مجموعة ثالثة تحتوي على ناتج تنفيذ عملية التقاطع بين مجموعتين.
- الطريقة insertElement تضيف العنصر k إلى المجموعة.
- الطريقة deleteElement تحذف العنصر m من المجموعة.
-الطريقة setPrint تطبع محتويات المجموعة وتطبع "Empty Set" إذا كانت المجموعة فارغة.
-الطريقة isEqualTo تقارن بين مجموعتين فيما إذا كانا متساويين أم لا.

اكتب برنامج يستخدم كائنات من نوع هذا الصنف.

المراجع

١. Deitel and Deitel, Java: How to Program, 3rd Edition, Prentice Hall, 2001
٢. Patrick Naughton and Michael Morrison, the Java Handbook, McGraw-Hill, 1996
٣. Bruce Eckel, Thinking in Java (2nd Edition), 2001
٤. م. فادي حجار، لغة البرمجة JAVA 2، دار شعاع للنشر والعلوم، ٢٠٠١
٥. كن أرنولد، لغة برمجة جافا، مركز التعريب والبرمجة، ٢٠٠١

المحتويات

الصفحة

تقدر المؤسسة العامة للتعليم الفني والتدريب المهني الدعم

المالي المقدم من شركة بي آيه إي سيستمز (العمليات) المحدودة

GOTEVOT appreciates the financial support provided by BAE SYSTEMS

BAE SYSTEMS



المملكة العربية السعودية

المؤسسة العامة للتعليم الفني والتدريب المهني

الإدارة العامة لتصميم وتطوير المناهج

البرمجيات

برمجة ٣

٢٤٣ حاب

مقدمة

الحمد لله وحده، والصلاة والسلام على من لا نبي بعده، محمد وعلى آله وصحبه، وبعد:

تسعى المؤسسة العامة للتعليم الفني والتدريب المهني لتأهيل الكوادر الوطنية المدربة القادرة على شغل الوظائف التقنية والفنية والمهنية المتوفرة في سوق العمل، ويأتي هذا الاهتمام نتيجة للتوجهات السديدة من لدن قادة هذا الوطن التي تصب في مجملها نحو إيجاد وطن متكامل يعتمد ذاتياً على موارده وعلى قوة شبابه المسلح بالعلم والإيمان من أجل الاستمرار قدماً في دفع عجلة التقدم التنموي: لتصل بعون الله تعالى لمصاف الدول المتقدمة صناعياً.

وقد خطت الإدارة العامة لتصميم وتطوير المناهج خطوة إيجابية تتفق مع التجارب الدولية المتقدمة في بناء البرامج التدريبية، وفق أساليب علمية حديثة تحاكي متطلبات سوق العمل بكافة تخصصاته لتلبي متطلباته ، وقد تمثلت هذه الخطوة في مشروع إعداد المعايير المهنية الوطنية الذي يمثل الركيزة الأساسية في بناء البرامج التدريبية، إذ تعتمد المعايير في بنائها على تشكيل لجان تخصصية تمثل سوق العمل والمؤسسة العامة للتعليم الفني والتدريب المهني بحيث تتوافق الرؤية العلمية مع الواقع العملي الذي تفرضه متطلبات سوق العمل، لتخرج هذه اللجان في النهاية بنظرة متكاملة لبرنامج تدريبي أكثر التصاقاً بسوق العمل، وأكثر واقعية في تحقيق متطلباته الأساسية.

وتتناول هذه الحقيبة التدريبية " برمجة ٣ " لمتدربي قسم" البرمجيات " للكليات التقنية موضوعات حيوية تتناول كيفية اكتساب المهارات اللازمة لهذا التخصص.

والإدارة العامة لتصميم وتطوير المناهج وهي تضع بين يديك هذه الحقيبة التدريبية تأمل من الله عز وجل أن تسهم بشكل مباشر في تأصيل المهارات الضرورية اللازمة، بأسلوب مبسط يخلو من التعقيد، وبالاستعانة بالتطبيقات والأشكال التي تدعم عملية اكتساب هذه المهارات.

والله نسأل أن يوفق القائمين على إعدادها والمستفيدين منها لما يحبه ويرضاه إنه سميع مجيب الدعاء.

الإدارة العامة لتصميم وتطوير المناهج

تمهيد



برمجة ٣

الوراثة و تعدد الأشكال

الجداره:

أن يكون المتدرب قادراً على فهم المبادىء الأساسية لبرمجة الكائنات مثل الوراثة وتعدد الأشكال والقدره على كتابة برامج تحتوي على فصائل جديدة ترث طرق وبيانات فصائل تم إنشاؤها وتنفيذها بصورة جيدة.

الأهـداف:

١. مراجعة المفاهيم الأساسية لبرمجة الكائنات في لغة الجافا
٢. تعلم مبادىء الوراثة
٣. فهم كيفية وراثة واستبدال طرق الفصائل العليا
٤. فهم فكرة تعدد الأشكال

مستوى الاداء المطلوب:

أن يصل المتدرب إلى اتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعده:

- وجود حاسب آلي
- وجود بيئة متكاملة لبناء وتنفيذ برامج لغة الجافا
- دفتر
- قلم

الوراثة وتعدد الأشكال

Inheritance and Polymorphism

مقدمة

مقرر برمجة ٢ تنأول المفاهيم الأساسية لبرمجة الكائنات Object Oriented programming وأن لغة الجافا تعتمد على هذه المفاهيم وفي هذه المقدمة نختصر بعض هذه المفاهيم لتكون اساساً للموضوعات المتقدمة في مجال برمجة الكائنات التي يتنأولها هذا المقرر.

من دراستنا السابقة للغة الجافا تبين أن البرنامج يحتوي على مجموعة من الفصائل Classes ويمكن إنشاء مجموعة من الكائنات Objects التي تأخذ خصائص الفصيلة المنشأة منها وأن كل فصيلة تحتوي على مجموعة من الطرق Methods التي تبين سلوك الكائنات وكذلك تحتوي الفصيلة على مجموعة من المتغيرات Instance variables التي تحتفظ بخصائص الكائن المنشئا من هذه الفصيلة ويبين شكل(١ - ١) مثال لبناء فصيلة لحساب البنك وسوف نقوم بشرح مكونات هذه الفصيلة لمراجعة المفاهيم الأساسية لبرمجة الكائنات في لغة الجافا وتكون أساساً لبناء المفاهيم المتطورة.

أولاً الفصيلة Class

الفصيلة هي القالب الذى نستخدمه في إنشاء الكائنات Objects وكل فصيلة لها خصائصها Attributes والتي تحددها البيانات Data أو المتغيرات Member variables ولها سلوكها Behaviorوهي الطرق Member Methods والشكل العام لتعريف فصيلة هو:

```
Access specifier        class           class_name
Ex:
public          class           BankAccount □
```

عند إنشاء برنامج بلغة الجافا يحتوي على العديد من الفصائل يمكن وضع كل فصيلة في ملف منفصل ويبداء بمحدد الوصول للفصيلة عام public أو تجميع الفصائل في ملف واحد ويبدأ تعريف الفصيلة التي تحتوي على الطريقة main بكلمة الوصول عام public class وتبدأ باقىالفصائل بالكلمة المحجوزة class فقط.

ثانياً المنشآت Constructors

تحتوي هذه الفصيلة على نوعين من المنشآت Constructors

١. المنشئا الأول وهو الإفتراضي ويعطى قيمة صفر للحساب أثناء إنشاء كائن جديد من هذه الفصيلة وهذا المنشأ لايحتوي اي عوامل Parameters داخل الأقواس ولكن تظل الأقواس مطلوبة

BankAccount.java

```
1  public class BankAccount
2  {
3  // The first constructor is the default constructor sets balance to zero
4           public BankAccount()
5           {
6                   balance = 0;
7           }
8  // The second constructor sets balance to initial to initial balance
9           public BankAccount(double initialBalance)
10          {
11                  balance = initialBalance;
12          }
13  // The deposit method adds an amount to instance variable balance
14          public void deposit(double amount)
15          {
16                  balance = balance + amount;
17          }
18  // The withdraw method subtracts an amount from instance variable balance
19          public void withdraw(double amount)
20          {
21                  balance = balance - amount;
22          }
23  // The transfer method withdraw an amount from this object and deposit to
other object balance
24          public void transfer(BankAccount other , double amount)
25          {
26                  withdraw(amount);
27                  other.deposit(amount);
28          }
29
30          // The getBalance method returns the current balance
31          public double getBalance()
32          {
33                  return balance;
34          }
35                  // The instance variable balance
36          private double balance;
37 }
```

شكل (١ -١)

٢. المنشأ الثانى ويعطى قيمة ابتدائية للحساب أثناء إنشاء كائن جديد من هذه الفصيلة ويحتوي بين الأقواس على عامل واحد من نوع البيانات الرقمية ذي الفصلة العشرية double

ومن الملاحظ في هذا المثال وفي لغة الجافا بصفة عامة أن

١. الغرض من المنشآت Constructors هو إعطاء قيم أولية لمتغيرات الحالة للكائن عند إنشائه أول مرة من الفصيلة
٢. المنشآت تأخذ نفس اسم الفصيلة
٣. الفصيلة يمكن أن تحتوي على العديد من المنشآت ويقوم المترجم بتحديد أي منهم يستدعى من خلال العوامل داخل الأقواس
٤. تكون الصيغة العامة للمنشأ في لغة الجافا كالتالي

```
Access Specifier      Class name (parameter type    parameter name, ….)
Ex:
Public                BankAccount (double           initialBalance)□
```

ثالثاً الطرق Methods

الفصيلة BankAccount تتكون من أربعة طرق Methods تمثل العمليات الأساسية في تعامل البنوك مع العملاء وهي

١ - عملية الإيداع deposite Method
٢ - عملية السحب withdraw method
٣ - عملية الاستعلام عن الرصيد getBalance method
٤ - عملية التحويل من حساب إلى حساب transfer method

الشكل العام لعنوان الطريقة method هو :

```
Access specifier      return type      method name(parameter type parameter name,..)
Ex 1:
public           void             deposite (double amount)
Ex 2:
public           double           getBalance
```

- المجال الأول في العنوان يبين محدد الوصول إلى الطريقة عام public
- المجال الثاني في العنوان يبين نوع البيانات العائدة بعد التنفيذ مثل double ولتبين أن الطريقة لاتسترجع أي بيانات نستخدم الكلمة void.

- المجال الثالث وهو اسم الطريقة وهو إختياري ومن الأفضل استخدام اسم يدل على وظيفة الطريقة ودائما في لغة الجافا يبدأ الاسم بحرف صغير small letter وفي حالة الاسم الذى يتكون من أكثر من مقطع يبدأ المقطع الأول بحرف صغير ثم المقاطع الآخرى بحرف كبير مثل getBalance
- المجال الرابع وهو الأقواس وفي داخلها معاملات الطريقة يفصل بينهما فصلة (,) وفي حالة عدم وجود عوامل تظل الأقواس مستخدمة وهذه العوامل تسمى عوامل ظاهرة explicit parameter
- هناك نوع آخر من العوامل يخص الطريقة وهو غير الظاهر(الضمني) implicit parameters هذا العامل الضمني من نوع الفصيلة التي تعرف الطريقة

رابعاً الكائنات Objects **وأحيانا تسمى** Instance

وهي عناصر تمثيل استخدام الفصيلة في البرنامج وهي تأخذ نفس شخصية الفصيلة من دوال وبيانات والطريقة الوحيدة لإنشاء الكائن بإستخدام المؤثر new

Ex1:

```
BankAccount myAccount = new BankAccount();
```

الجملة السابقة في لغة الجافا تقوم بإنشاء كائن يسمى myAccount ويستدعى المنشىء الأول الإفتراضي BankAccount() في الفصيلة BankAccount ويعطى قيمة أولية صفر لنسخة متغير الكائن balance للكائن myAccount

Ex2:

```
BankAccount m1 = new BankAccount(5000);
```

هذه الجملة في لغة الجافا تقوم بإنشاء كائن يسمى m1 وتستدعي المنشىء الثاني BankAccount(double initialBalance) من الفصيلة BankAccount ويعطى قيمة أولية ٥٠٠٠ ريال لنسخة متغير الكائن balance للكائن m1

خامساً متغيرات الكائنات instance variables

كما رأينا في الأمثلة السابقة لإنشاء الكائنات أن كل كائن يخزن حالته في واحد أو أكثر من متغيرات الكائن ويكون الشكل العام لتعريف متغير الكائن هو:

```
Access specifier        type        variable name;
Ex:
private             double        balance;
```

المجال الأول ويعرف بمحدد الوصول للمتغير وبصفة عامة نستخدم المحدد خاص private لمتغيرات الكائنات وهذا يعني أنه يمكن الوصول لهذه المتغيرات فقط بدوال نفس الفصيلة المعرف فيها المتغير ولايمكن تغييره من أى دالة آخرى ولذلك نستطيع القول بأن متغير الكائن غير ظاهر للمبرمج الذى يستخدم الفصيلة المعرف فيها وهذه العملية من إخفاء البيانات data hiding تسمى تغليف Encapsulation وهي المبدأ الأول من مبادىء برمجة الكائنات OOP

ونلاحظ أن كل كائن له نسخته من المتغيرات

```
Ex1:
myAccount.balance
Ex2:
m1.balance
```

فعندما تشير إلى متغير فيطريقة فإنك تشير تلقائياً إلى متغير الكائن المستخدم في استدعاء الطريقة وفي هذه الحالة يكون الكائن هو المتغير الضمنى للطريقة

```
Ex:
m1.deposit(500);
```

عند تنفيذ هذه الجملة في لغة الجافا تقوم باستدعاء الطريقة deposit وتضيف ٥٠٠ ريال إلى حساب الكائن m1 وكأنه ينفذ الجملة في الطريقة deposit كالتإلي

```
balance = balance + amount
this.balnce = this.balance + amount
m1.balance = m1.balance +amount
```

الوراثة Inheritance

الوراثة inheritance هي المبدىء الثاني من مبادىء برمجة الكائنات OOP والتي يمكن الإستفادة منها في لغة جافا لتطوير البرامج حيث يمكن استخدام الفصائل التي تم تصميمها وتنفيذها وتأكدنا من أنها تعمل بصورة جيدة ثم نكتب فصيلة جديدة يضاف اليها الطرق والبيانات الجديدة فقط وترث الطرق والبيانات الموجودة في الفصيلة القديمة التي يمكن اعتبار الفصيلة الجديدة امتداداً لها

مثال : نفترض أننا نريد إنشاء حساب بنكي يضيف عائداً شهرياً على الرصيد الموجود في البنك.

بدراسة هذا النوع من الحساب يتضح لنا أن هذا الحساب هو امتداد لحساب البنك التي تم برمجته في الفصيلة BankAccount حيث إنه يتطلب عمليات ايداع وسحب واستعلام عن رصيد وله متغير لتخزين نسخة من قيمة الحساب ولذلك يمكن استخدام مبدأ الوراثة لتطوير البرنامج السابق بإنشاء فصيلة جديدة ويضاف إليها دالة لحساب العائد ويضاف إليها أيضا متغير لمعدل العائد الشهري. الشكل (١ -٢) يبين بناء هذه الفصيلة.

SavingAccount.java

```
/* The SavingAccount class extends the BankAccount class implements
   a new method addInterest to model an account that pays a fixed
   interest rate on deposits
*/
public class SavingAccount extends BankAccount
// The SavingAccount constructor
{
          public SavingAccount(double rate)
          {
                    interestRate = rate;
          }
// addInterest method
          public void addInterest()
          {
                    double interest = getBalance()*interestRate/100;
                    deposit(interest);
          }
// The SavingAccount instance variable
          private double interestRate;
}
```

شكل (١ -٢)

وبتحليل هذا المثال يمكن أن نستعرض بعض خصائص الوراثة في لغة جافا ومنها

١ - من أسباب استخدام الوراثة هو إعادة استخدام شفرة البرنامج code reuse حيث يمكن استخدام فصائل موجودة ونوفر الجهد المبذول لإتقان تصميم وتنفيذ هذه الفصائل.

٢ - الفصيلة التي تورث تسمى الفصيلة العامة (العليا) superclass لأنها تحتوي على الطرق والبيانات المشتركة وأحيانا تسمى فصيلة الأب parent class أو الفصيلة الأساسية Base class

٣ - الفصيلة التي ترث تسمى الفصيلة الفرعية subclass لأنها تحتوي على الطرق والبيانات الخاصة المضافة وأحيانا تسمى فصيلة الابن child class أو الفصيلة المشتقة drived class

٤ - ولتحقيق عملية الوراثة وتوريث فصيلة قديمة إلى فصيلة جديدة عند إنشائها نقوم بكتابة اسم الفصيلة الجديدة ثم الكلمة المحجوزة الدالة على الوراثة extends التي تعني أن هذه الفصيلة هي امتداد للفصيلة القديمة التي يكتب اسمها بعدها وتكون الصيغة العامة للوراثة هي:

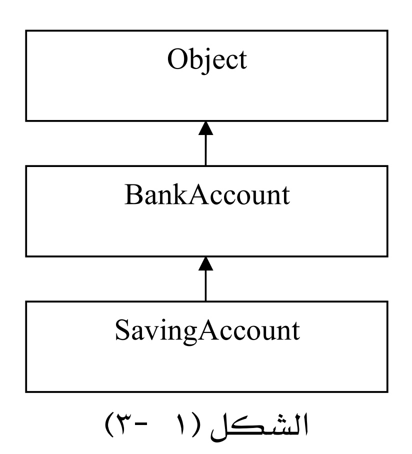
```
class subclass Name        extends        superclass Name
ex:
class SavingAccount        extends        BankAccount
```

وهذا يعني أن في المثال السابق الفصيلة BankAccount هي الفصيلة العامة (العليا) superclass وأن الفصيلة SavingAcount هي الفصيلة الفرعية subclass

٥ - عندما نقوم بإنشاء فصيلة ولم نحدد اسم فصيلة ترث منها تفترض لغة جافا أنك ترث من الفصيلة الأم Object ومثال ذلك الفصيلة BankAccount ترث الفصيلة Object

٦ - الفصيلة Object تحتوي على عدد صغير من الطرق التي تعني شيئاً لجميع الكائنات مثل الطريقة toString التي يمكن استخدامها للحصول على وصف حالة الكائن.

الشكل (١ -٣) يبين علاقة الوراثة بين الفصائل الثلاثة Object و BankAccount و SavingAccount و يسمى مخطط الوراثة Inheritance Diagram



الشكل (١ -٣)

٧ - عند إنشاء كائن جديد من الفصيلة الفرعية ينادى أولاً المنشىء الموجود في الفصيلة العامة ليعطي قيمة مبدئية للمتغيرات الموجودة فيها ثم ينفذ المنشىء الموجود في الفصيلة الفرعية لإعطاء قيم أولية للمتغيرات الجديدة

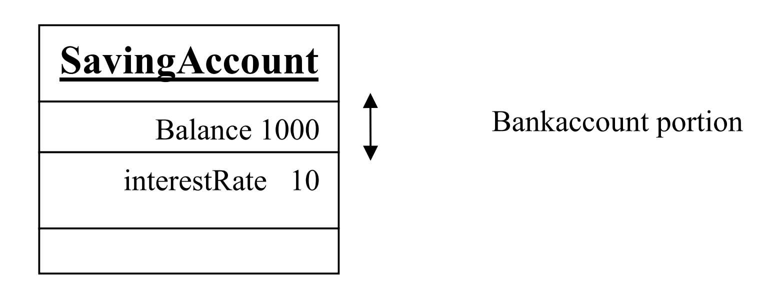
Ex:

```
SavingAccount          m2 = new SavingAccount(10)
```

هذه الجملة في لغة الجافا تقوم بإنشاء كائن m2 من نوع حساب البنك SavingAccount وتستدعي المنشىء الافتراضي في الفصيلة العامة BankAccount لإعطاء قيمة صفر للمتغير balance ثم تستدعى المنشىء الموجود في الفصيلة Saving Account لإعطاء قيمة أولية ١٠ للمتغير intrestRate وبذلك تكون نسخة متغيرات الكائن بعد تنفيذ هذه الجملة كلأتي :

```
m2.balance = 0
m2.intrestRate = 10
```

أي أن الكائن من الفصيلة الفرعية ورث المتغير balance من الفصيلة العامة وأضيف إليه المتغير intrestRate من فصيلته ويبين الشكل (١ -٤) مثال للكائن m2 من الفصيلة SavingAccount



شكل (١ -٤)

٨ - الكائن المنشىء من الفصيلة الفرعية يعتبر حالة خاصة من الفصيلة العامة ويمكنه استدعاء الطرق الموجودة فيها كما ينادي الطرق الموجودة في الفصيلة الفرعية.

Ex: m2.addIntrest();

هذه الجملة في لغة جافا تستدعي الطريقة addIntrest من الفصيلة الفرعية SavingAccount التي يتم فيها حساب العائد على الحساب الحالي وإضافته على الحساب وتلاحظ في هذه الطريقة أنها تستدعي الطرق getBalance و deposit من الفصيلة العامة BankAccount لأن الفصيلة الفرعية ترث الطرق من الفصيلة العامة وبسبب عدم وجود كائن أثناء استدعاء هذه الطرق فإنها تستخدم المتغير الضمني الذي يستدعي الطريقة addIntrest وتنفذ جمل الطريقة addIntrest كالآتى :

```
double   interest =this.getBalance() * this.intrestRate/100
this.deposit(intrest)
```

وبتنفيذ الجملة ;()m2.addIntrest يكون التنفيذ لجمل الطريقة ()addIntrest كالأتى:

```
double   interest =  m2.getBalance() * m2.intrestRate/100
m2.deposit(intrest)
```

٩ - يجب أن نعلم أن أي فصيلة لها فصيلة عليا واحدة فقط ولكن الفصيلة العليا يمكن أن يكون لها أي عدد من الفصائل الفرعية ولذلك يطلق على لغة جافا أنها تستخدم single inheritance وذلك عكس لغة ++C التي يمكن فيها أن يكون لها أكثر من فصيلة عليا ويطلق على هذا النوع من الوراثة Multiple inheritance وعلى الرغم من أن هذا النوع يمثل قوة في البرمجة إلا أنه يسبب الكثير في التعقيدات من التصميم وتتبع هيكل الفصائل.

المخطط الهرمي للوراثة Inheritance Hierarchies

المخطط الهرمي عادةً يمثل كشجرة حيث معظم الفصائل التي تمثل المفاهيم العامة قريبة من الجزر root والفصائل الأكثر تخصصية نحو الفروعbranches وسوف نستخدم مثال بسيط للمخطط الهرمي لإستكمال دراسة مفاهيم الوراثة.

المثال : نفترض أن البنك يريد أن يقدم لعملائه ثلاثة أنواع من الحسابات:

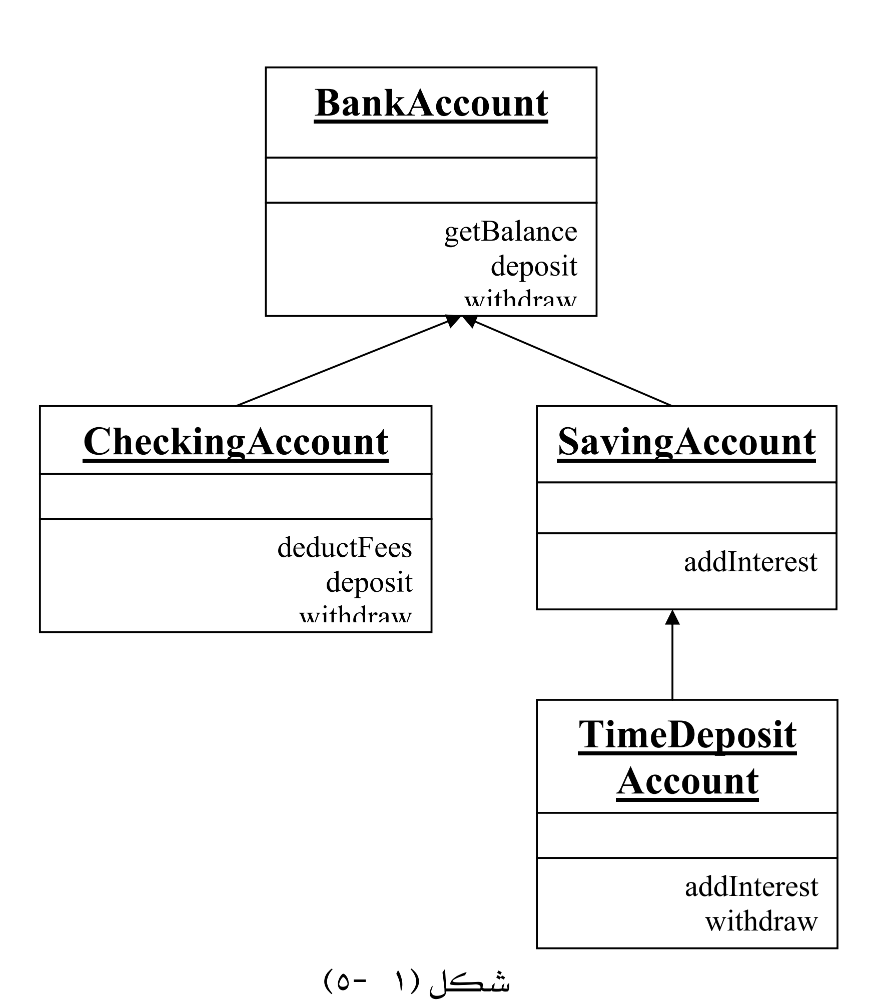
١ - الحساب الأول CheckingAccount ليس له عائد ويسمح للعميل بعدد قليل من العمليات البنكية كل شهر ثم يلزمه بدفع أجرة عن كل عملية إضافية.

٢ - الحساب الثاني SavingAccount يعطي عائداً مركباً شهرياً

٣ - الحساب الثالث TimeDepositAccount يعطى عائد مركب شهرياً ولكن يلتزم العميل بترك المال في الحساب لعدد معين من الشهور بدون سحب ويوجد شرط جزائي للسحب المبكر.

بتحليل هذه الحسابات نجد أنها حسابات بنكية تشترك جميعها في المتغير balance وكذلك في عمليات الإيداع deposit والسحب withdraw و االاستعلام عن الرصيد getBalance ولذلك يمكن القول أنها جميعاً

يمكـن أن تـرث الفصيلة BankAccount وكـذلك يمكـن القـول بـأن الحسـاب الثـاني والثالـث متشـابهان ومختلفإن عن الحساب الأول ولذلك يمكن تمثيل المخطط الهرمي للوراثة لهذه الفصائل البنكية كما هو موضح في الشكل (١ -٥)



شكل (١ -٥)

الطرق methods **ومتغيرات الكائنات** instance variables **للفصائل الفرعية**

أولاً الطرق Methods :

عند تعريف طرق فصيلة فرعية يمكن أن يكون هناك ثلآثة احتمالات

١ - استخدام الطريقة في الفصيلة الفرعية بنفس الاسم ونفس المعاملات (أي نفس البصمة) كما في الفصيلة العامة ولكن هناك استبدال للجمل التنفيذية لهذه الطريقة وتسمى override method ولذلك عند استدعاء هذه الطريقة باستخدام متغير من نوع الفصيلة الفرعية فيتم تنفيذ الطريقة الموجودة في الفصيلة الفرعية وليست الطريقة الأصلية الموجودة في الفصيلة العليا.

٢ - يمكن للفصيلة الفرعية أن ترث الطرق الموجودة في الفصيلة العليا بدون أى تغير وفي هذه الحالة عند استدعاء هذه الطرق باستخدام كائن من نوع الفصيلة الفرعية يتم تنفيذ الطريقة من الفصيلة العليا.

٣ - يمكن للفصيلة الفرعية إنشاء طرق جديدة لتحقيق الغرض من التطوير وتستدعى هذه الطرق الجديدة فقط باستخدام كائنات الفصيلة الفرعية.

ثانياً متغيرات الكائنات Instance variables

بالنسبة لمتغيرات الكائنات يوجد حالتان فقط

١ - جميع متغيرات الكائنات للفصيلة الفرعية تورث تلقائياً لكائنات الفصيلة الفرعية

٢ - يمكن تعريف متغير جديد يطبق فقط على كائنات الفصيلة الفرعية

ويمكن تطبيق هذه الاحتمالات على الحساب الأول فقد تم إنشاء فصيلة جديدة تسمى CheckingAccount كما في شكل (١ -٦) وهذه الفصيلة ترث فصيلة BankAccount وفيها تم إضافة طريقة جديدة deductFees() لحساب الأجرة الشهرية وأيضا تم إضافة متغير كائن transactionCount لتتبع عدد العمليات الشهرية ونجد أيضاً أنه تم استبدال الطرق deposit و withdraw لزيادة أعداد العمليات transactionCount أثناء عمليات السحب ولايداع

الكائن المنشىء من الفصيلة CheckingAccount يكون له متغيران

١ - الأول balance يرثه من الفصيلة BankAccount

٢ - الثاني transactionCount معرف جديد في فصيلة CheckingAccount

ويطبق على الكائن المنشىء من الفصيلة CheckingAccount أربعة طرق وهي :

١ - الاستعلام عن الرصيد getBalance() يرثها من فصيلة BankAccount

٢ - الإيـداع (deposit(double amount (تستبدل الطريقة الموجودة في الفصيلة BankAccount)
٣ - السحب (withdraw(double amount (تستبدل الطريقة الموجودة في الفصيلة BankAccount
٤ - الطريقة ()deductFees معرفة جديدة في فصيلة CheckingAccount

والآن دعنا نشرح تنفيذ دالة الإيداع (deposit(double amount في الفصيلة CheckingAccount لنبين كيفية تغير متغير كائن في الفصيلة العليا وكيفية استدعاء دالة من الفصيلة العليا لها نفس البصمة في الفصيلة الفرعية.

CheckingAccount.java

```
/* The CheckingAccount class extends the BankAccount class implements
   a new method deductFees and overide the deposit and withdraw methods
*/
public class CheckingAccount extends BankAccount
// The CheckingAccount constructor
{
            public CheckingAccount(double initialBalance)
            {
                        // construct superclass
                        super(initialBalance);
                        // initialize transaction count
                         transactionCount = 0;
            }
// overide the BankAccount deposit method
            public void deposit(double amount)
            {
                        transactionCount ++;
                        // now add amount to balance
                        super.deposit(amount);
            }
// overide the BankAccount withdraw method
            public void withdraw(double amount)
            {
                        transactionCount ++;
                        // now subtract amount from balance
                        super.withdraw(amount);
            }
// New method deductFees
            public void deductFees()
            {
                        if(transactionCount>FREE_TRANSACTIONS)
                                    {           double fees =
TRANSACTION_FEE*(transactionCount- FREE_TRANSACTIONS);
                                                super.withdraw(fees);
                                    }
                        transactionCount = 0;

            }
// The CheckingAccount instance variables
            private int transactionCount;
            private static final int FREE_TRANSACTIONS = 3;
            private static final double TRANSACTION_FEE = 2.0;
}
```

شكل (١ -٦)

طريقة الإيداع في الفصيلة الفرعية هي مستبدلة override من الفصيلة العليا فلها نفس الاسم والعوامل والغرض منها هو زيادة أعداد العمليات وإيداع المبلغ فتكون بالصورة الآتية

```
// override the BankAccount deposit method
        public void deposit(double amount)
        {
                transactionCount ++;
                // now add amount to balance
                balance = balance + amount;          //ERROR
        }
```

تنفيذ الإيداع بهذه الطريقة خطأ لأن محدد الوصول للمتغير balance معرف في الفصيلة العليا أنه خاص private ولذلك فإن دوال الفصيلة الفرعية ليست لها الأحقية في تغيير البيانات الخاصة في الفصيلة العليا ولذلك يجب استخدام دوال الفصيلة العليا لتعديل متغيرات الكائن فيها فإذا استخدمنا الطريقة deposit(amount) من الفصيلة العامة لتغيير قيمة المتغير balance يصبح بناء الطريقة في الفصيلة الفرعية كالآتى

```
// override the BankAccount deposit method
        public void deposit(double amount)

                transactionCount ++;
                // now add amount to balance
                deposit(amount);
```

هنا نجد مشكلة آخرى وهي أن الطريقة deposit(amount) تستدعى بالمتغير الضمني this وهو من نوع الفصيلة CheckingAccount التي تملك دالة بنفس البصمة deposit(double amount) ولذلك فإن تنفيذ البرنامج يدخل دائرة مغلقة إلى مالانهاية والحل الصحيح لهذه المشكلة أن لغة جافا تستخدم الكلمة المحجوزة super لتحديد استدعاء الطريقة من الفصيلة العليا فتكون الصيغة الصحيحة لتنفيذ دالة الإيداع في الفصيلة الفرعية هي:

```
// override the BankAccount deposit method
        public void deposit(double amount)

                transactionCount ++;
                // now add amount to balance
                super.deposit(amount);
```

وبذلك يمكن تنفيذ دالة السحب بنفس الطريقة فتكون الصيغة الصحيحة لها هي

```
// override the BankAccount withdraw method
        public void withdraw(double amount)
```

```
            transactionCount ++;
        // now subtract amount from balance
        super.withdraw(amount);
```

تنفيذ الطريقة الجديدة deductFees

تستدعى هذه الطريقةنهاية كل شهر لتنفيذ جملة شرطية لإختبار إذا كان عدد عمليات الكائن أكبر من عدد العمليات المجانية فإذا تحقق الشرط تقوم الطريقة بحساب الأجر fees المطلوب على العمليات الزائدة وخصم الأجرة من حساب الكائن باستخدام الطريقة withdraw(fees) وقبل نهاية الطريقة تقوم بإعطاء قيمة صفر لعداد العمليات لبدء عمليات شهر جديد والصيغة الصحيحة لهذه الطريقة هي

```
// New method deductFees
public void deductFees()

    if(transactionCount>FREE_TRANSACTIONS)

    double fees = TRANSACTION_FEE*(transactionCount- FREE_TRANSACTIONS);
    super.withdraw(fees);

    transactionCount = 0;
```

تنفيذ الحساب الثالث TimeDepositAccount

هذا النوع من الحساب هو امتداد للنوع الثاني SavingAccount لأنه يقوم بحساب العائد الشهري على الإيداع ولكن الاختلاف في هذا الحساب أن المودع يعد بترك المال لعدة شهور بدون سحب مقابل زيادة في معدل العائد ويتم خصم قيمة جزائية إذا تم سحب قبل انتهاء المدة ويبين الشكل (١ -٧) تنفيذ الفصيلة TimeDepositAccount وفيها يتم تعريف متغير كائن جديد periodsToMaturity لتحديد عدد شهور الإيداع ويتم إعطاء قيمة لهذا المتغير عند إنشاء كائن من هذه الفصيلة ويتم إنقاص هذا العداد في نهاية كل شهر عند حساب العائد في الطريقة addIntrest

من المخطط الهرمي شكل(١ -٥) يجب ملاحظة أن الفصيلة TimeDepositAccount تبعد مستويين عن الفصيلة BankAccount ولكن تظل الفصيلة BankAccount فصيلة عليا ولكنها ليست الفصيلة العليا المباشرة للفصيلة TimeDepositAccount ولكنها ترث منها الطريقتين getBalance و deposit

ولذلك يمكن القول أنه يمكن توريث الطرق من فصيلة عليا غير مباشرة بشرط أن لايكون تم استبدالها في الفصائل الفرعية بينهما.

تنفيذ طرق الفصيلة TimeDepositAccount

في هذه الفصيلة يوجد طريقتان

الطريقة الأولى addInterest() وهي طريقة مستبدلة override method للطريقة الموجودة في الفصيلة العليا SavingAccount لحساب العائد الشهري وإنقاص عدد شهور فترة الإيداع فيكون تنفيذ الطريقة كالتإلى

```
// override the SavingAccount addInterest method
        public void addInterest()

                periodsToMaturity--;
                super.addInterest();
```

الطريقة الثانية withdraw(double amount) وفيها تستخدم جملة شرطية لاختبار عدد شهور الإيداع إذا كان أكبر من الصفر تقوم الطريقة بخصم القيمة الجزائية ثم سحب القيمة المطلوبة وإن لم يتحقق الشرط تقوم بسحب القيمة فقط ولذلك يكون تنفيذ الطريقة كالتإلي:

```
// override the BankAccount withdraw method
        public void withdraw(double amount)

                if (periodsToMaturity >0)
                        super.withdraw(EARLY_WITHDRAW_PENALITY);
                // now subtract amount from balance
                super.withdraw(amount);
```

TimeDepositAccount.java

```
/* The TimeDepositAccount extends SavingAccount class overrides
   the SavingAccount addInterest method and the BankAccount withdraw
   method to model an account like SavingAccount but you promize to leave
   the mony in the account for a particular number of months, and
   there is a penalty for early withdrawal.
*/
public class TimeDepositAccount extends SavingAccount
// The TimeDepositAccount constructor
{
        public TimeDepositAccount(double rate , int maturity)
        {
                super (rate);
                periodsToMaturity = maturity;
        }
// override the SavingAccount addInterest method
        public void addInterest()
        {
                periodsToMaturity--;
                super.addInterest();
        }
        // override the BankAccount withdraw method
        public void withdraw(double amount)
        {
                 if (periodsToMaturity >0)
                        super.withdraw(EARLY_WITHDRAW_PENALITY);
                // now subtract amount from balance
                super.withdraw(amount);
        }
// The TimeDepositAccount instance variables
        private int periodsToMaturity;
        private static final double EARLY_WITHDRAW_PENALITY = 20.0;
}
```

شكل (١ -٧)

أثناء تنفيذ طرق الفصيلة TimeDepositAccount لاحظ أن هذه الطرق تستخدم الجملة super.addIntrest(); و الجملة super.withdraw(amount); لإستدعاء الطرق من الفصيلة العليا. بالنسبة للطريقة addIntreset تستدعي من الفصيلة العليا المباشرة SavingAccount ولكن بالنسبة للطريقة withdraw فإن الفصيلة العليا المباشرة لاتملك الطريقة withdraw ولذلك تورث هذه الطريقة وتستبدل من الفصيلة العليا غير المباشرة BankAccount وبصفة عامة تستدعى الطرق من الفصيلة الأقرب في المخطط الهرمي للوراثة.

منشآت الفصيلة الفرعية Subclass constructors

عند إنشاء كائن من الفصيلة CheckingAccount يجب إعطاؤه قيمة للمتغير balance ولذلك يجب استدعاء منشىء الفصيلة العليا BankAccount لإعطاء قيمة للمتغير balance ولذلك يجب استخدام الكلمة المحجوزة super ويتبعها معاملات المنشىء بين أقواس ويجب أن تكون هذه الجملة هي الجملة الأولى في منشىء الفصيلة الفرعية ويكون تنفيذ منشىء الفصيلة الفرعية كالتالي:

```
// The CheckingAccount constructor

        public CheckingAccount(double initialBalance)

                // construct superclass
                super(initialBalance);
                // initialize transaction count
                 transactionCount = 0;
```

إذا لم يستدع منشىء الفصيلة الفرعية منشىء الفصيلة العامة فإن المنشىء الإفتراضى للفصيلة العامة يستدعى تلقائياً وإن لم يكن هناك منشىء افتراضي في الفصيلة العامة فإن المترجم يعطي خطأ ويمكن تنفيذ منشىء الفصيلة الفرعية CheckingAccount بدون استدعاء منشىء الفصيلة العليا BankAccount وحينئذ يستدعى المنشىء الإفتراضى للفصيلة BankAccount ويعطى قيمة صفر للمتغير balance وبعد ذلك يقوم منشىء الفصيلة الفرعية CheckingAccount بإيداع القيمة الأولية للمتغير balance ويكون تنفيذ المنشىء كالتإلي:

```
// The CheckingAccount constructor

        public CheckingAccount(double initialBalance)

        // superclass has been constructed with its default //constructor
        // now set initial balance
                super.deposit(initialBalance);
                // initialize transaction count
                 transactionCount = 0;
```

ولكن في حالة الفصيلة TimeDepositAccount ليس لنا اختيار حيث إن الفصيلة العليا SavingAccount ليس لها منشىء افتراضي ولذلك يجب استدعاء منشىء الفصيلة العليا كالتإلي

```
// The TimeDepositAccount constructor

        public TimeDepositAccount(double rate , int maturity)

                super (rate);
                periodsToMaturity = maturity;
```

تعدد الأشكال Polymorphism

علاقة الوراثة أحياناً تسمى "is – a" كل كائن من الفصيلة الفرعية يكون أيضاً كائناً من الفصيلة العليا ولكن بخصائص خاصة وهذا يعنى أن كل كائن من الفصيلة ChechingAccount هو كائن من الفصيلة BankAccount ولذلك يمكن استخدام كائن من الفصيلة الفرعية مكان كائن من الفصيلة العليا

مثال ذلك الطريقة transfer في فصيلة BankAccount تستخدم لتحويل مبلغ من المال من حساب إلى حساب آخر وتنفيذ هذه الطريقة كالتإلي

```
// The transfer method withdraw an amount from this object and //deposit to other object balance
        public void transfer(BankAccount other , double amount)

                withdraw(amount);
                other.deposit(amount);
```

وحيث إن جميع فصائل البنك هي امتداد للفصيلة BankAccount يمكن تمرير أي كائن من أي حساب للطريقة transfer

```
Ex:
    BankAccount c1 = new BankAccount(1000);
    CheckingAcount c2 = new CheckingAccount(2000);
    C1.transfer(c2,500);
```

دعنا نتتبع استدعاء هذه الطريقة بدقة. داخل استدعاء هذه الطريقة يوجد متغيرين من نوع BankAccount ولكنهما يشيرا إلى كائن BankAccount وكائن CheckingAccount كما في شكل(١ -٨) ولنأخذ في اعتبارنا استدعاء الطريقة deposit أي طريقة deposit المعامل other من نوع فصيلة Bankaccount ولذلك فإن الظاهر هو استدعاء الطريقة BankAccount.deposit ومن ناحية آخرى نجد أن الفصيلة CheckingAccount تملك الطريقة deposit التي تحدث متغير أعداد العمليات transactionCount وحيث إن المتغير other في الطريقة transfer فعليا يشير كائن الفصيلة الفرعية CheckingAccount يكون من المناسب لو أن الطريقة CheckingAccount.depoit تستدعى بدلاً من الطريقة BankAccount.deposit

وفي لغة جافا استدعاء الطريقة دائماً يحدد بنوع المتغير الفعلي actual object وليس نوع المرجعية للمتغير reference object وهذا يعني أن نفس الجملة other.deposit يمكن أن تنادي طرق مختلفة وهذا المبدأ أو الأساس يسمى تعدد الأشكال وهو المبدأ الثالث من مبادىء برمجة الكائنات OOP. والتعبير polymorphism يأتى من الكلمات الإغريقية many shapes

في لغة الجافا جميع طرق الكائنات instance methods تكون متعددة الأشكال polymorphic

في لغة الجافا أيضاً يمكن أن يكون هناك عدة طرق في فصيلة واحدة لها نفس الاسم ولكن المعاملات الظاهرة explicit parameters للطرق مختلفة وهذا ما يسمى بطرق التحميل الذائد overloaded methods ومثال ذلك المنشآت دائماً لها نفس الاسم ولكن تختلف المعاملات وحينئذ يختار المترجم compiler الطريقة المناسبة عند ترجمة البرنامج طبقاً لتطابق المعاملات الظاهرة explicit parameters للطريقة المستدعاة
هناك فرق بين تعدد الأشكال polymorphism والتحميل الزائد overload وهو أن في حالة التحميل الزائد overload المترجم يحدد الطريقة أثناء ترجمة البرنامج أي قبل تنفيذ البرنامج وهذا الاختيار للطريقة يسمى رابط مبكر early binding ولكن في حالة اختيار الطريقة التي تطابق نوع المعامل الضمني كما في حالة الطريقة deposit التي تم تحليلها سابقاً فإن المترجم compiler لايأخذ أي قرار عند ترجمة البرنامج ويتم تنفيذ البرنامج قبل أن يعرف ماذا يخزن في المتغير other ولذلك فإن الآلة التخيلية virtual machine وليس المترجم compiler هي التي تختار الدالة المناسبة وهذا الاختيار يسمى الرابط المتآخر late binding

البرنامج BankAccountTest في شكل (١ -٨) و البرنامج polymorphismTest في شكل(١ -٩) هي برامج تطبيقية يجب على القارىء تتبع تنفيذها يدوياً قبل استعراض نتائجها من الحاسب لتأكيد المفاهيم التي تم شرحها في هذه الوحدة.

BankAcountTest.java

```
public class BankAcountTest
{
        /* This program is used to test the diferent bankAcounts             classes
        */
        // The main method
        public static void main(String [] args)
        {
                BankAccount mohamed = new BankAccount();
                BankAccount ahmed = new BankAccount(1000);
                BankAccount mahmoud = new BankAccount(2000);
                mohamed.deposit(3000);
                ahmed.deposit(3000);
                mahmoud.deposit(3000);
                // transfer 1000Sr from mahmoud to ahmed
                mahmoud.transfer(ahmed,1000);
                System.out.println(mohamed.getBalance());
                System.out.println(ahmed.getBalance());
                System.out.println(mahmoud.getBalance());
                // construct new SavingAccount object called moustafa
                SavingAccount moustafa = new SavingAccount(5);
                moustafa.deposit(1000);
                moustafa.addInterest();
                System.out.println(moustafa.getBalance());
        // construct new CheckingAccount object called baker
                CheckingAccount baker = new CheckingAccount(5000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.withdraw(2000);
                baker.deductFees();
                System.out.println(baker.getBalance());
        // construct new TimeDepositAccount object called flah
                TimeDepositAccount flah = new   TimeDepositAccount(5,5);
                flah.deposit(10000);
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.deposit(1000);
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.withdraw(5000);
                System.out.println(flah.getBalance());
```

```
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.withdraw(5000);
                System.out.println(flah.getBalance());
        }
}
```

شكل (١ -٨)

PolymorphismTest.java

```
public class PolymorphismTest
{
        /* This program is used to test the principles of polymorphism

        */


        // The main method

        public static void main(String [] args)
        {
                // construct new SavingAccount object called abdalah
                SavingAccount abdalah = new SavingAccount(0.5);

                // construct new TimeDepositAccount object called flah
                TimeDepositAccount flah = new TimeDepositAccount(1,3);

                // construct new CheckingAccount object called baker
                CheckingAccount baker = new CheckingAccount(0);

                abdalah.deposit(10000);

                flah.deposit(10000);

                printBalance("abdalah ",abdalah);
                printBalance("flah    ",flah);
                printBalance("baker   ",baker);

                // transfer 1000Sr from abdalah to baker

                abdalah.transfer(baker,2000);

                // transfer 1000Sr from flah to baker

                flah.transfer(baker,980);
```

```
            printBalance("abdalah ",abdalah);
            printBalance("flah    ",flah);
            printBalance("baker   ",baker);


            baker.withdraw(500);
            printBalance("baker   ",baker);
            baker.withdraw(80);
            printBalance("baker   ",baker);
            baker.withdraw(400);
            printBalance("baker   ",baker);

            endOfMonth(abdalah);
            endOfMonth(flah);
            endOfMonth(baker);
            printBalance("abdalah ",abdalah);
            printBalance("flah    ",flah);
            printBalance("baker   ",baker);
      }
      public static void endOfMonth(SavingAccount savings)
      {savings.addInterest();
      }
      public static void endOfMonth(CheckingAccount checking)
      {checking.deductFees();
      }
      public static void printBalance(String name , BankAccount account)
      {System.out.println("The balance of " + name +
       "account is " + account.getBalance()+"  SR");
      }
}
```

شكل (١ -٩)

نتائج تنفيذ البرنامج PolymorphismTest

```
The balance of abdalah account is 10000.0  SR
The balance of flah    account is 10000.0  SR
The balance of baker   account is 0.0  SR
The balance of abdalah account is 8000.0  SR
The balance of flah    account is 9000.0  SR
The balance of baker   account is 2980.0  SR
The balance of baker   account is 2480.0  SR
The balance of baker   account is 2400.0  SR
The balance of baker   account is 2000.0  SR
The balance of abdalah account is 8040.0  SR
The balance of flah    account is 9090.0  SR
The balance of baker   account is 1996.0  SR
```



المملكــة العربيــة الســعودية

المؤسسة العامة للتعليم الفني والتدريب المهني

الإدارة العامة لتصميم وتطوير المناهج

برمجة ٣

معالجة الإستثناءات

معالجة الاستثناءات

٢

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات وأنواع الاستثناءات في لغة الجافا والقدرة على كتابة برامج تحتوي على معالجة هذه الاستثناءات

الأهـداف:

١. فهم أساسيات معالجة الاستثناءات في لغة الجافا
٢. تعلم أنواع الاستثناءات
٣. استخدام التعليمة try لاحتواء جزء البرنامج الذي يمكن أن يحدث به استثناء
٤. استخدام التعليمة catch لمعالجة الأنواع المختلفة للاستثناءات
٥. استخدام التعليمة finally
٦. القدرة على تمرير استثناء

مستوى الأداء المطلوب:

أن يصل المتدرب إلى اتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- وجود بيئة متكاملة لبناء وتنفيذ برامج لغة الجافا
- دفتر
- قلم

معالجة الاستثناءات

Exception Handling

مقدمة :

الاستثناء هو مؤشر لحدوث خطأ أثناء عملية تنفيذ البرنامج مما يؤدي إلى تعطيل التسلسل الطبيعي لتعليمات البرنامج وقد تعلمنا في الفصل السابق أن الوراثة في لغة الجافا تعطيها صفة الامتدادية وهذه الصفة يمكن أن تزيد من عدد ونوع الأخطاء التي يمكن أن تحدث حيث إن كل فصيلة جديدة تضاف إلى البرنامج يمكن أن تضيف مصدراً من مصادر الاستثناءات في البرنامج. إذاً نستطيع القول أن الاستثناء هو حدوث خطأ ما وهذا الخطأ ليس خطأ في بناء الجملة syntax error ولكنه قد يكون له العديد من المصادر مثل القسمة على صفر ومعاملات غير متاحة للدالة و الإشارة إلى عنصر في المصفوفة خارج نطاقها.

عند حدوث استثناء يحتاج البرنامج إلى معالجة هذا الاستثناء لكي يستمر تنفيذ البرنامج بصورة طبيعية وسابقا قبل عام ١٩٩٠ كانت معالجة الاستثناءات تتم باختبار قيم صحيحة تعود بدلائل مثل القيمة صفر تدل على النجاح والقيمة السالبة تدل على نوع من الاستثناءات وهذه القيم أصبحت تعرف بشفرات الأخطاء Error codes وقد تم اكتشاف أن استخدام هذا النوع من معالجة الأخطاء يتسبب في ثلاث مشاكل:

١ - غالباً تهمل شفرة الخطأ

٢ - اختبار شفرة الأخطاء تعترض التدفق الطبيعي للبرنامج مما يصعب تتبع المستخدم للبرنامج

٣- اختبار شفرة الأخطاء يزيد من حجم البرنامج

أساسيات معالجة الاستثناء في لغة الجافا

The basics of java Exception Handling

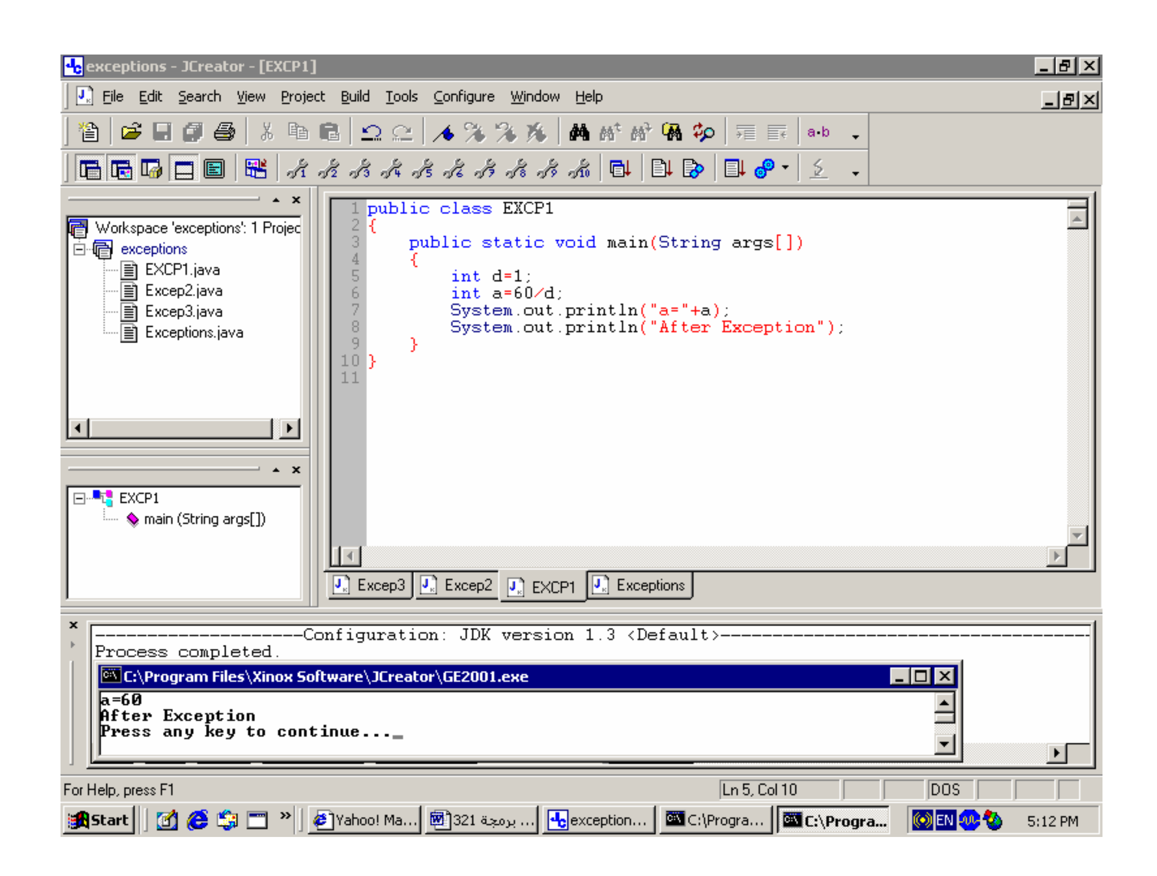
لقد أدت مشاكل استخدام شفرة الأخطاء Error codes إلى تطوير آلية جديدة لمعالجة الاستثناءات في لغة الجافا تعتمد على الكائنات مما أدى إلى برامج سهلة القراءة والتتبع وكذلك برامج أكثر مرونة.

وفي هذا النموذج عند حدوث استثناء أثناء تشغيل برنامج الجافا إما البرنامج program أو آلة لغة الجافا الافتراضية JVM تنشىء كائن لوصف الاستثناء ويشمل هذا الكائن قيم المتغيرات في لحظة حدوث الاستثناء.

إذا تم إنشاء الكائن من البرنامج فإن البرنامج يمرر ذلك الكائن إلى آلة الجافا الافتراضية JVM وعند استقبال الكائن تبحث في البرنامج عن معالج الاستثناء handler exception الذى يمكن أن يعالج الاستثناء الموصوف بالكائن. إذا وجد المعالج يتم تمرير الكائن لمعالج الاستثناء الذي يقوم باستخدام محتويات الكائن لمعالجة الاستثناء. إذا لم يوجد معالج الاستثناء يتوقف البرنامج عن التنفيذ.

مثال ١

شكل (٢ -١) يبين برنامج بسيط لقسمة رقمين إذا تتبعنا هذا البرنامج نجد أنه يتم تنفيذه بطريقة صحيحة لأنه يوجد قيمة غير صفرية للمقام (d)

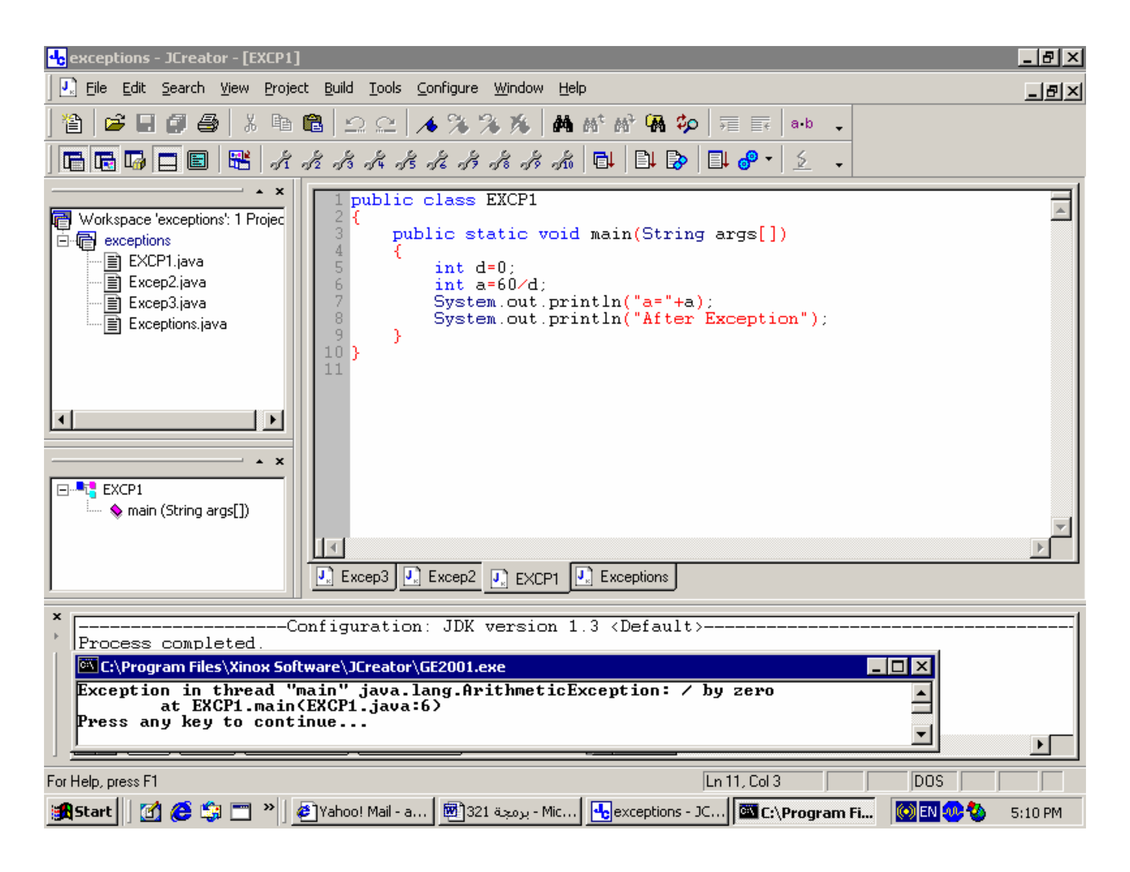


شكل (٢-١)

ولكن شكل (٢ -٢) يبين نفس البرنامج ولكن تم إعطاء المقام قيمة صفرية ولذلك اذا تتبعنا الشكل نجد أن المترجم (compiler) قد أنهى ترجمة البرنامج بنجاح وهذا ملاحظ من الجملة Process completed ولكن عند تنفيذ البرنامج هناك خطأ تنفيذي (استثناء) وهو القسمة على صفر فقد بحث عن معالج للخطأ في البرنامج فلم يجد ولذلك أعطى رسالة

Exception in thread "main" java.lang.ArithmeticException: /by zero at Excp1.main <Excp1.java:6>

وهذا يبين أن كائن الاستثناء من الفصيلة ArithmaticException ويشمل بيان الخطأ وهو القسمة على صفر by zero/ وقد بحث عن معالج داخل البرنامج فلم يجد وتلاحظ في شكل (٢ -٢) إنهاء تنفيذ البرنامج قبل تنفيذ جمل الطباعة



شكل (٢-٢)

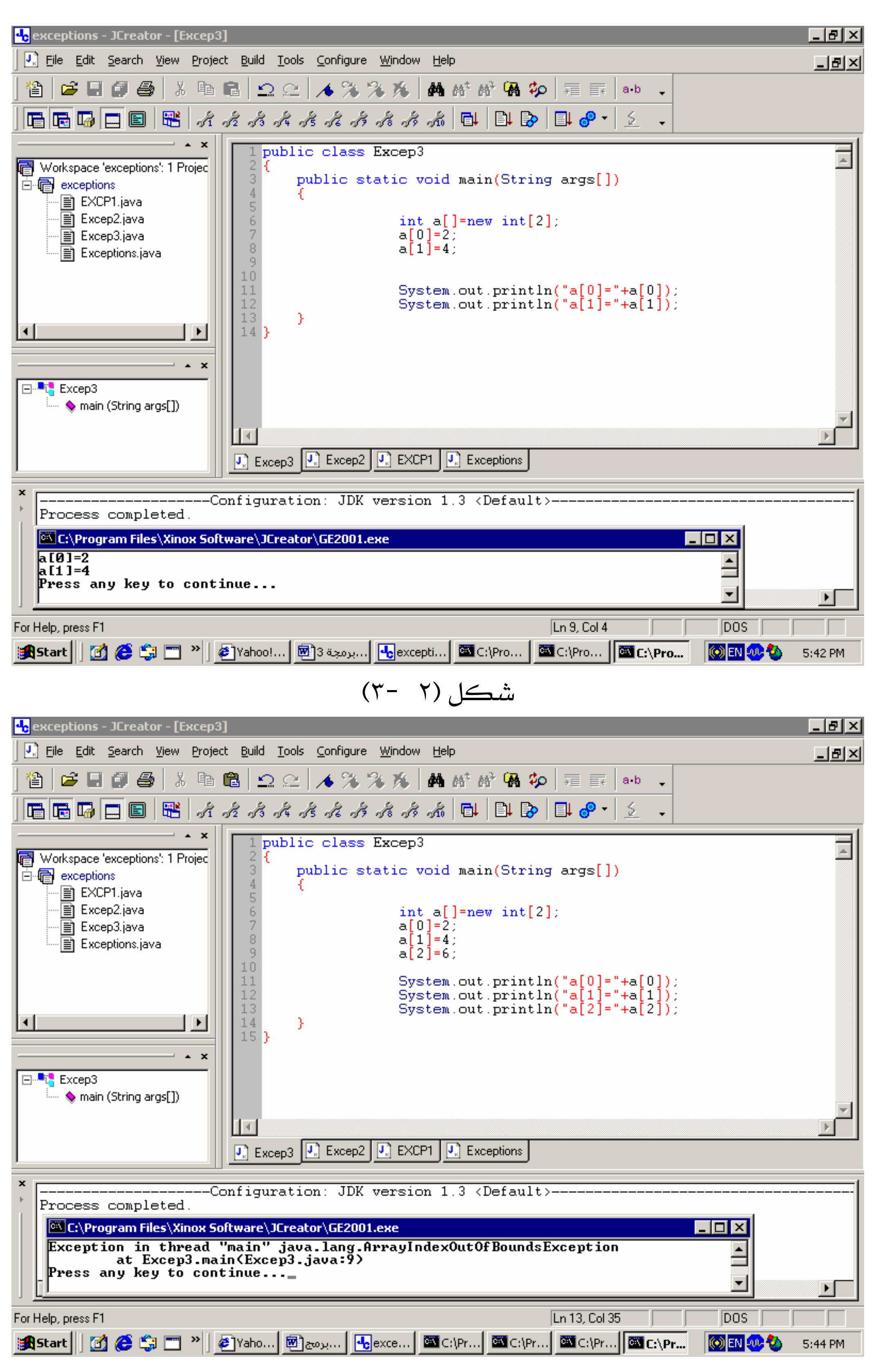
مثال ٢

يبين شكل (٢ -٣) برنامج بسيط لتعريف مصفوفة مكونة من عنصرين من نوع الأرقام الصحيحة وإعطاء قيم للعنصرين وطباعة قيم العنصرين تلاحظ من الشكل تنفيذ البرنامج بصورة طبيعية وتم إعطاء نتائج الطباعة.

ولكن شكل (٢ -٤) يبين نفس البرنامج مع إضافة الجملة a[2] = 6; ونتيجة أن هذا العنصر خارج نطاق تعريف المصفوفة فقد وقع استثناء وتم البحث عن معالج داخل البرنامج فلم يوجد فتوقف تنفيذ البرنامج قبل تنفيذ جمل الطباعة وأعطى رسالة الخطأ التالية:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundException at Excp1.main <Excp3.java:9>

وهذا يبين أن كائن الاستثناء من الفصيلة ArrayIndexOutOfBoundException


exceptions - JCreator - [Excep3]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'exceptions': 1 Projec
exceptions
EXCP1.java
Excep2.java
Excep3.java
Exceptions.java
Excep3
main (String args[])
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
System.out.println("a[0]="+a[0]);
System.out.println("a[1]="+a[1]);
}
}
Excep3 Excep2 EXCP1 Exceptions
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
a[0]=2
a[1]=4
Press any key to continue...
For Help, press F1
Ln 9, Col 4
DOS
Start
5:42 PM
exceptions - JCreator - [Excep3]
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
System.out.println("a[0]="+a[0]);
System.out.println("a[1]="+a[1]);
System.out.println("a[2]="+a[2]);
}
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
at Excep3.main(Excep3.java:9)
Press any key to continue...
For Help, press F1
Ln 13, Col 35
DOS
Start
5:44 PM

شكل (٢) -٣

شكل (٢) -٤

أنواع الاستثناءات Exception types

من الأمثلة السابقة تبين أن هناك العديد من أنواع الاستثناءات ومن معرفتنا للغة الجافا بأنها تتكون من فصائل فإن الاستثناءات في الجافا هي فصائل classes وكل فصيلة class تختص بنوع من الاستثناءات وجميع هذه الفصائل ترث الفصيلة العليا Throwable وتوجد فصيلتان فرعيتان ترثان هذه الفصيلة وهما Exception subclass و Error subclass وهذه الفصائل موجودة في الحزمة java.lang وهذه الفصائل الفرعية تصنف الاستثناءات أهي ذات علاقة بالبرنامج program related أم هي ذات علاقة بآلة الجافا الافتراضية JVM

الفصيلة Exception هي الفصيلة الجزرية root class لجميع الفصائل التي تصف جميع الاستثناءات ذات العلاقة ببرنامج جافا ويبين الجدول (٢ -١) بعض الفصائل الفرعية للفصيلة Exception ووصف كل منها.

الجدول (٢ -١) بعض الفصائل الفرعية للفصيلة Exception

الوصف	الفصيلة
الكائن المنشيء من هذه الفصيلة يصف مشكلة حسابية مثل محأولة القسمة على صفر	ArithmaticException
الكائن المنشيء من هذه الفصيلة يصف محأولة عنصر في مصفوفة بدليل غير معرف	ArrayIndexOutOfBoundsException
الكائن المنشيء من هذه الفصيلة يصف محأولة تحميل ملف فصيلة غير موجودة	ClassNotFoundException
الكائن المنشيء من هذه الفصيلة يصف محأولة ملف غير موجود	FileNotFoundException
الكائن المنشيء من هذه الفصيلة يصف خطأ عاماً أثناء عملية إدخال أو آخراج وتوجد فصائل فرعية من هذه الفصيلة لتصف الخطأ بدقة	IOException
الكائن المنشيء من هذه الفصيلة يصف محأولة استدعاء طريقة متغير كائن ليس له مرجعية لأي كائن null reference	NullPointerException

الفصيلة Error هي الفصيلة الجزرية root class لجميع الفصائل التي تصف الاستثناءات ذات العلاقة بآلة الجافا الافتراضية JVM مثال ذلك الفصيلة OutOfMemoryError class فإن الكائن المنشيء من هذه الفصيلة يصف محأولة خاطئة لتخصيص ذاكرة.

معالجة الاستثناءات في الجافا Exception Handling In java

تتم معالجة الأخطاء في لغة الجافا باستخدام مجموعة من التعليمات وهي:

a- try block
b- catch blocks
c- finally block
d- throw statement
e- throws clause

أولاً : try…catch finally blocks

تستخدم لغة الجافا التعليمة try لتحديد الجزء من البرنامج الذي يحتمل أن يحدث به خطأ ويجب أن يتبع هذا الجزء مباشرةً تعليمة catch أو أكثر لتحديد طريقة معالجة الأنواع المتوقعة من الاستثناءات ثم يتبع آخر تعليمة catch تعليمة finally وهي اختيارية وتستخدم لتحديد جزء من البرنامج يجب تنفيذه بغض النظر هل حدث استثناء في جزء تعليمة try أم لم يحدث استثناء ويكون الشكل العام للتعامل مع الاستثناءات في لغة الجافا كالتإلي:

```
try

        // Tested statements

catch ( ExceptionType 1   exob1)

        // exception handler for ExceptionType1

catch ( ExceptionType 2   exob2)

        // exception handler for ExceptionType2

finally

        // Statements that must be executed
```

مثال ٣

شكل (٢ -٥) يبين نفس البرنامج قسمة رقمين الذى تم شرحه في المثال ١ وفي هذا الشكل يتم معالجة الاستثناء ArithmaticException الذي حدث عندما أعطى المقام d قيمة صفرية ويتم ذلك باستخدام التعليمة try لإحتواء الجمل التي تسببت في الاستثناء كالتإلي:

```
try

        int d=0;
        int a=60/d;
```

ويتبع ذلك تعليمة catch التي تعالج هذا النوع من الاستثناء كالتالي:

```
catch (ArithmeticException   e)

            System.out.println("divide by zero");
            System.out.println(e.getMessage());
```

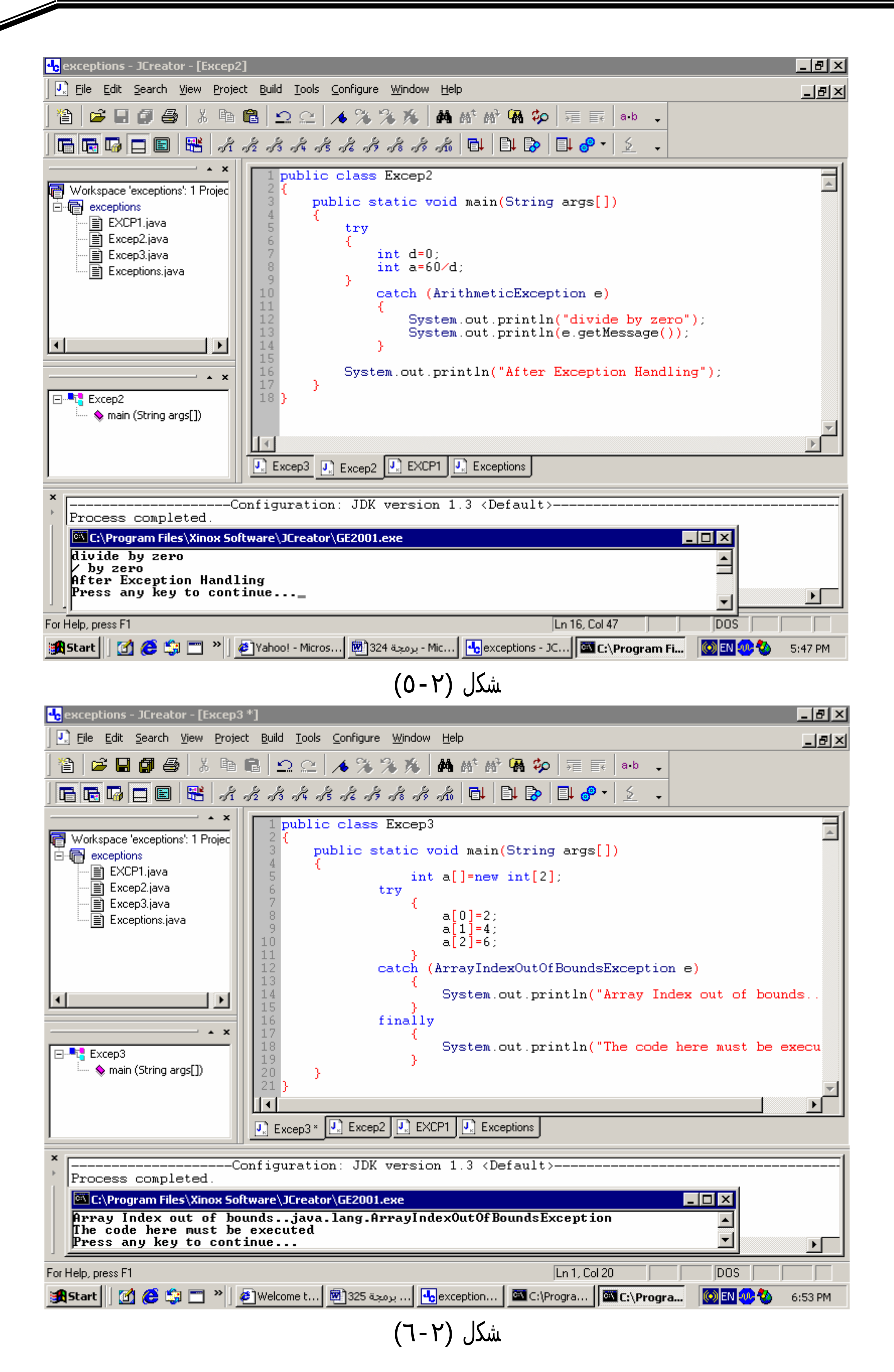
ويمكن تتبع تنفيذ البرنامج حيث تم حدوث استثناء عند تنفيذ السطر رقم ٨ في شكل (٢ -٥) وهو محأولة القسمة على صفر فتم إنشاء كائن من الفصيلة ArithmeticException وتم البحث عن معالج لهذا الاستثناء في البرنامج فوجد معالج مطابق للإستثناء في سطر ١٠ فتم انتقال تسلسل تنفيذ البرنامج إلى معالج الاستثناء الذي يحتوي على جملتين الأولى في السطر ١٣ لإظهار الرسالة divide by zero والثانية في السطر ١٤ لإظهار الرسالة التي يحتويها كائن الاستثناء e وهي by zero/ وبعد تنفيذ المعالج تم تنفيذ الجملة التي تتبع المعالج مباشرةً في سطر ١٦ لإظهار الرسالة After Exception Handling وتم إنهاء البرنامج بصفة طبيعية.

مثال ٤

شكل (٢ -٦) يبين نفس برنامج المصفوفة الذي تم شرحه في المثال ٢ وفي هذا الشكل يتم معالجة الاستثناء ArrayIndexOutOfBoundsException الذي حدث عند محأولة إعطاء عنصر المصفوفة a[2] قيمة ٦ في سطر ١٢ ويتم التقاط الاستثناء باستخدام التعليمة try لاحتواء الجمل التي تسببت في الاستثناء كالتالي:

```
try

            a[0]=2;
            a[1]=4;
            a[2]=6;
```


exceptions - JCreator - [Excep2]
public class Excep2
{
public static void main(String args[])
{
try
{
int d=0;
int a=60/d;
}
catch (ArithmeticException e)
{
System.out.println("divide by zero");
System.out.println(e.getMessage());
}
System.out.println("After Exception Handling");
}
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
divide by zero
/ by zero
After Exception Handling
Press any key to continue...
exceptions - JCreator - [Excep3 *]
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
try
{
a[0]=2;
a[1]=4;
a[2]=6;
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("Array Index out of bounds..
}
finally
{
System.out.println("The code here must be execu
}
}
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Array Index out of bounds..java.lang.ArrayIndexOutOfBoundsException
The code here must be executed
Press any key to continue...

شكل (٢-٥)

شكل (٢-٦)

ولمعالجة الاستثناء ArrayIndexOutOfBoundsException نستخدم تعليمة catch كالتالي:

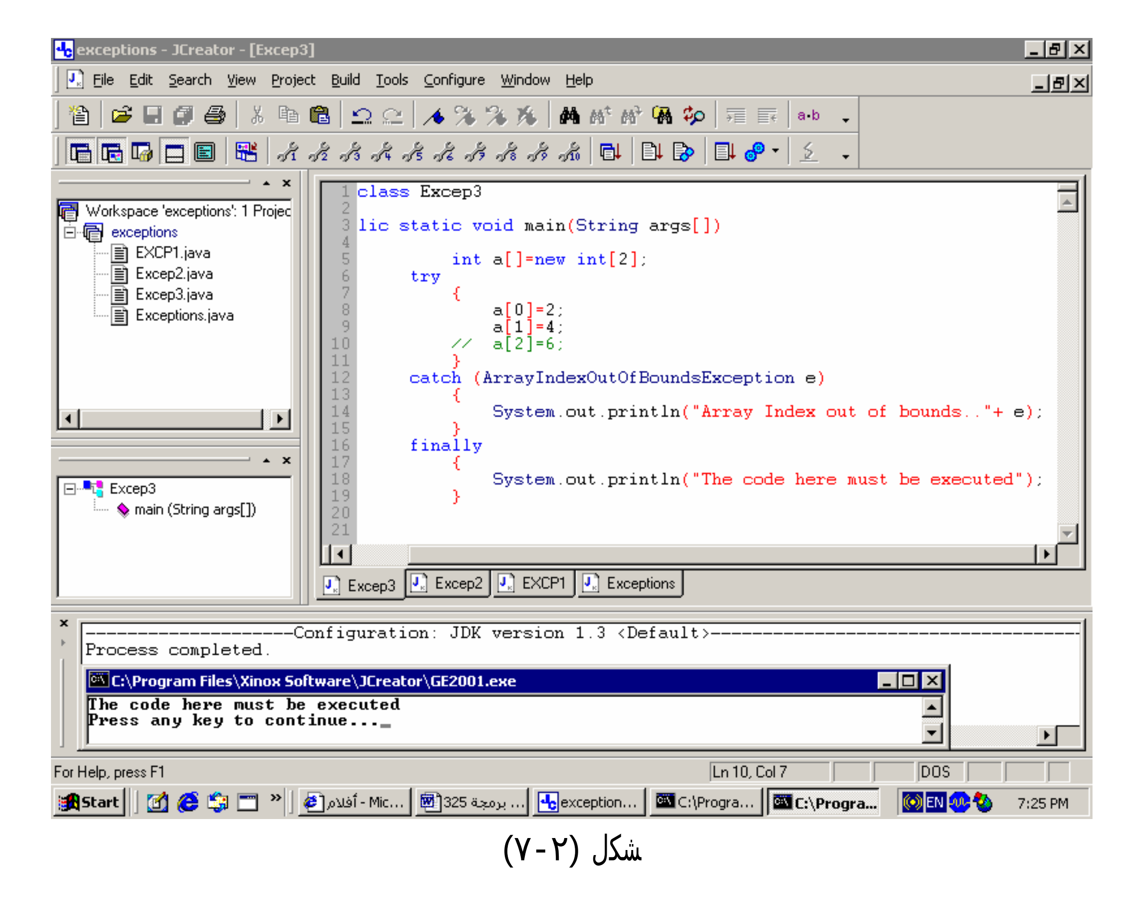
catch (ArrayIndexOutOfBoundsException e)

System.out.println("Array Index out of bounds.."+ e);

وفي هذا البرنامج تم استخدام تعليمة finally لإضاح كيف يمكن احتواء جزء البرنامج الذي يجب تنفيذه بعد تعليمة try سواء حدث استثناء أم لا.

وبتتبع تنفيذ هذا البرنامج في شكل (٢ -٦) نجد أنه تم تنفيذه بطريقة صحيحة حيث تم التقاط الاستثناء عند تنفيذ السطر ١٠ وتم البحث عن معالج الاستثناء وبدأ تنفيذه من السطر ١٢ وفيه تم إظهار الرسالة Array Index out of bounds بالإضافة إلى الذي يحتويها هو الكائن(e) وهو java.lang.ArrauIndexOutofBoundsException ثم بعد ذلك تم تنفيذ جزء البرنامج الذى تحتويه التعليمة finally وهو إظهار الرسالة :

The code here must be executed



شكل (٢-٧)

شكل (٢ -٧) يبين تنفيذ البرنامج السابق بعد إلغاء الجملة التي تسببت في الاستثناء فنجد أنه تم تنفيذ الجملة داخل التعليمة finally كما ذكرنا سابقا أنه يتم تنفيذ هذا الجزء من البرنامج سواء كان هناك استثناء أم لا.

تعدد تعليمة catch

عند توقع حدوث أكثر من استثناء في جزء من البرنامج فيمكن احتواء هذا الجزء بتعليمة try لالتقاط الأنواع المختلفة من الاستثناءات ثم نتبع ذلك بالعديد من تعليمات catch كلاً منها يعالج نوعاً من أنواع الاستثناءات المتوقعة.

مثال ٥

شكل (٢ -٨) يبين برنامج يجمع بين الجمل التي تستخدم لقسمة رقمين و الجمل التي تستخدم لتعريف مصفوفة وإعطاء قيم لعناصرها وفي هذا البرنامج تم إضافة التعليمة try لاحتواء هذه الجمل من البرنامج ثم اتبع ذلك بتعليمتين catch الأولى لمعالجة الاستثناء ArithmeticException والثانية لمعالجة الاستثناء ArrayIndexOutOfBoundsException .

ولتتبع تنفيذ هذا البرنامج نجد حدوث استثناء عند تنفيذ جملة القسمة في سطر ٨ وتم التقاط هذا النوع من الاستثناء والبحث له عن معالج فإنتقل تسلسل البرنامج إلى السطر ١٥ لتنفيذ المعالج وتم طباعة الرسائل

Handling the first Exception divide by zero
/ by zero

ثم انتقل تسلسل تنفيذ البرنامج إلى التعليمة finally وتم إظهار الرسالة

The code here must be executed

ملحوظة هامة: في المثال السابق يجب ملاحظة أن تسلسل تنفيذ البرنامج لم يعد مرة آخرى إلى جزء تعليمة try في السطر ٩ الذي يلي نقطة انتقال التسلسل للبحث عن معالج الاستثناء.

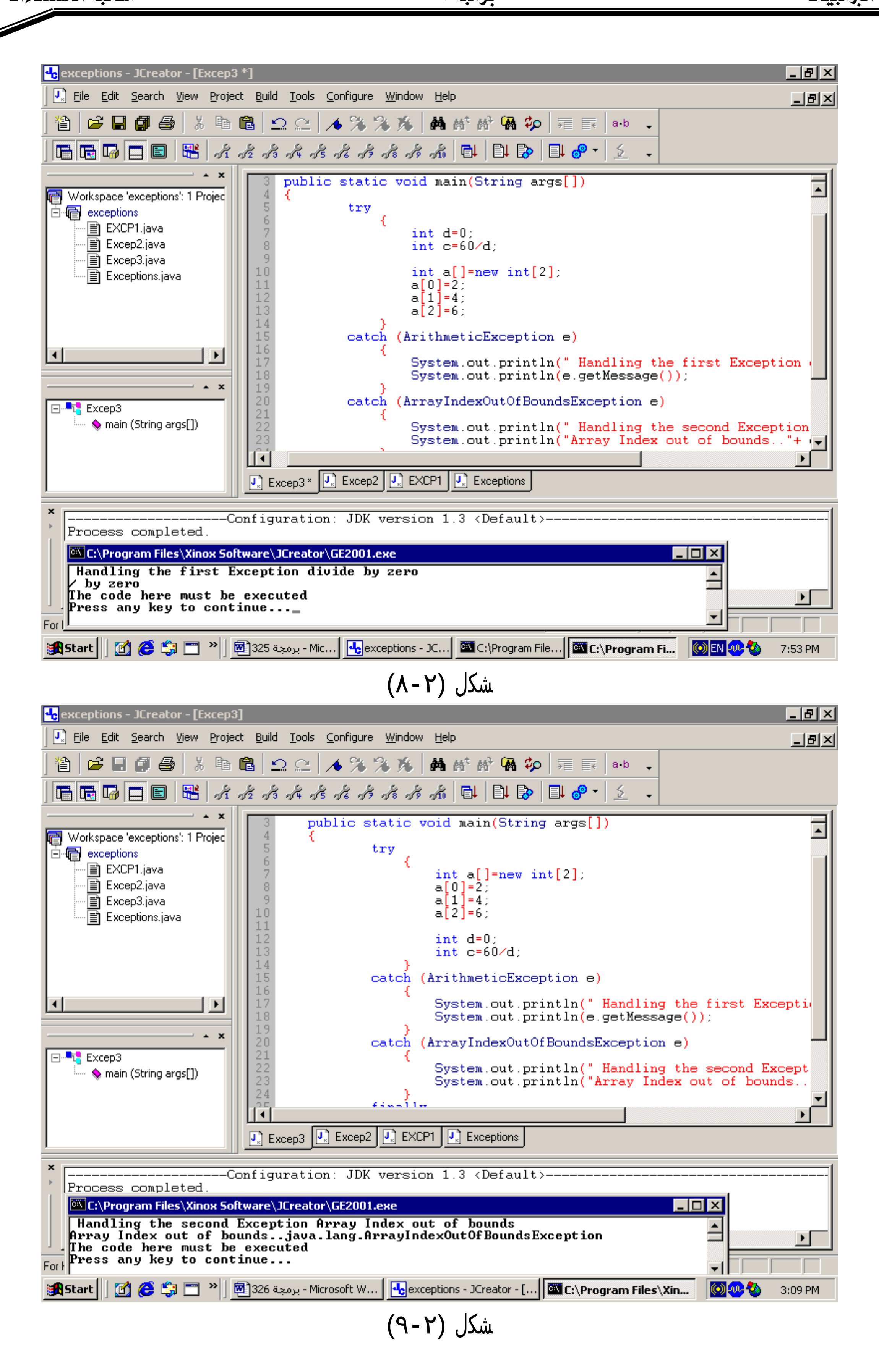
شكل (٢-٩) يبين نفس البرنامج مع تقدم جمل المصفوفة عن جمل قسمة رقمين ولذلك عند تنفيذ البرنـامج حـدث استثناء في السطر ١٠ وتم انتقال تسلسل البرنامج إلى السطر ٢٠ وإظهار الرسائل

Handling the second Exception Array Index out of bounds
Array Index out of bounds… java.lang.ArrauIndexOutofBoundsException

ثم تم الانتقال إلى تعليمة finally لإظهار الرسالة

The code here must be executed

ولم يعد تسلسل البرنامج إلى تعليمة try مرة آخرى.

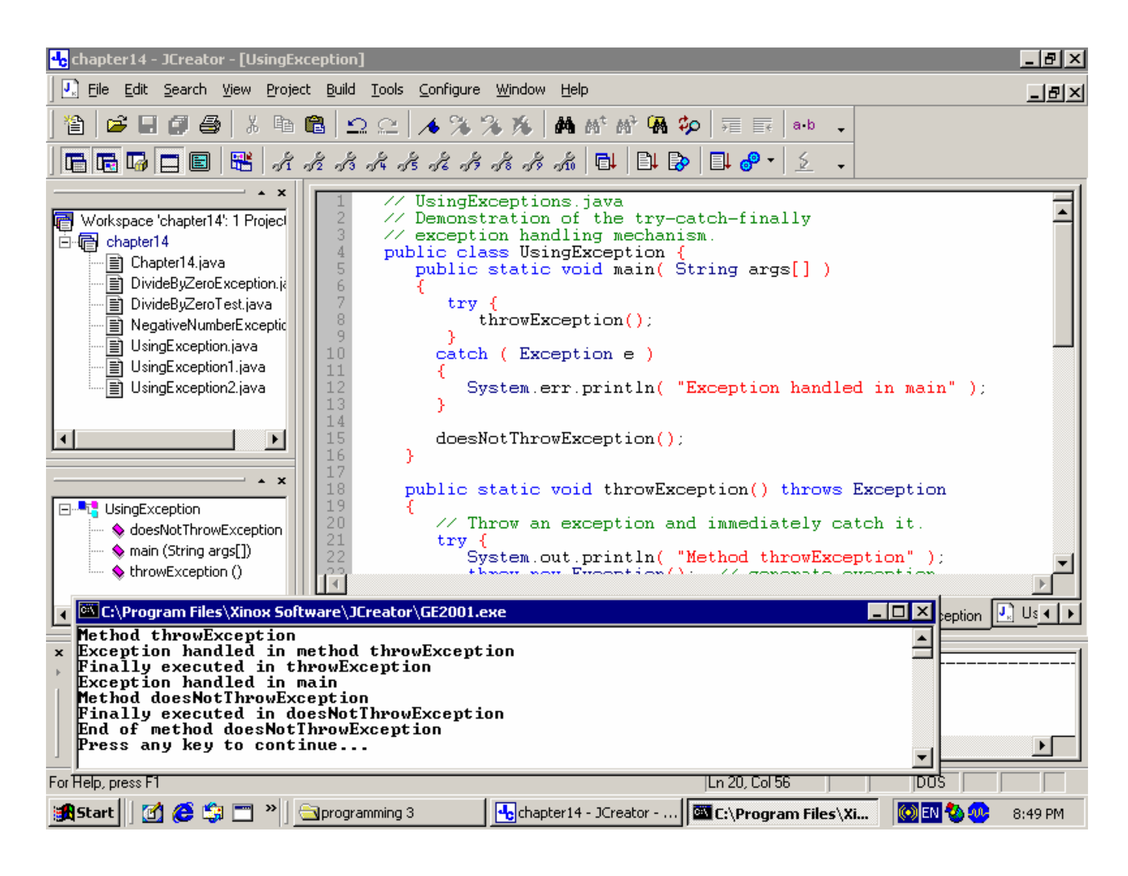


exceptions - JCreator - [Excep3 *]
public static void main(String args[])
{
try
{
int d=0;
int c=60/d;
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
}
catch (ArithmeticException e)
{
System.out.println(" Handling the first Exception
System.out.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println(" Handling the second Exception
System.out.println("Array Index out of bounds.."+
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Handling the first Exception divide by zero
/ by zero
The code here must be executed
Press any key to continue...

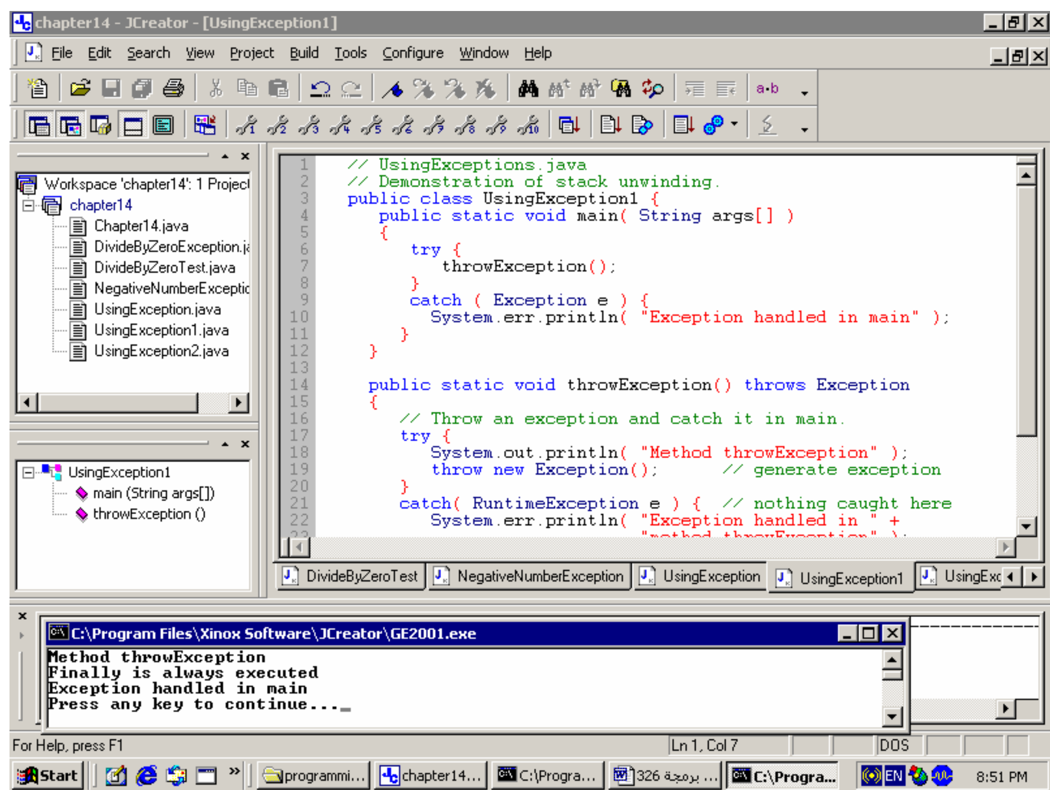
شكل (٢-٨)

exceptions - JCreator - [Excep3]
public static void main(String args[])
{
try
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
int d=0;
int c=60/d;
}
catch (ArithmeticException e)
{
System.out.println(" Handling the first Excepti
System.out.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println(" Handling the second Except
System.out.println("Array Index out of bounds..
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Handling the second Exception Array Index out of bounds
Array Index out of bounds..java.lang.ArrayIndexOutOfBoundsException
The code here must be executed
Press any key to continue...

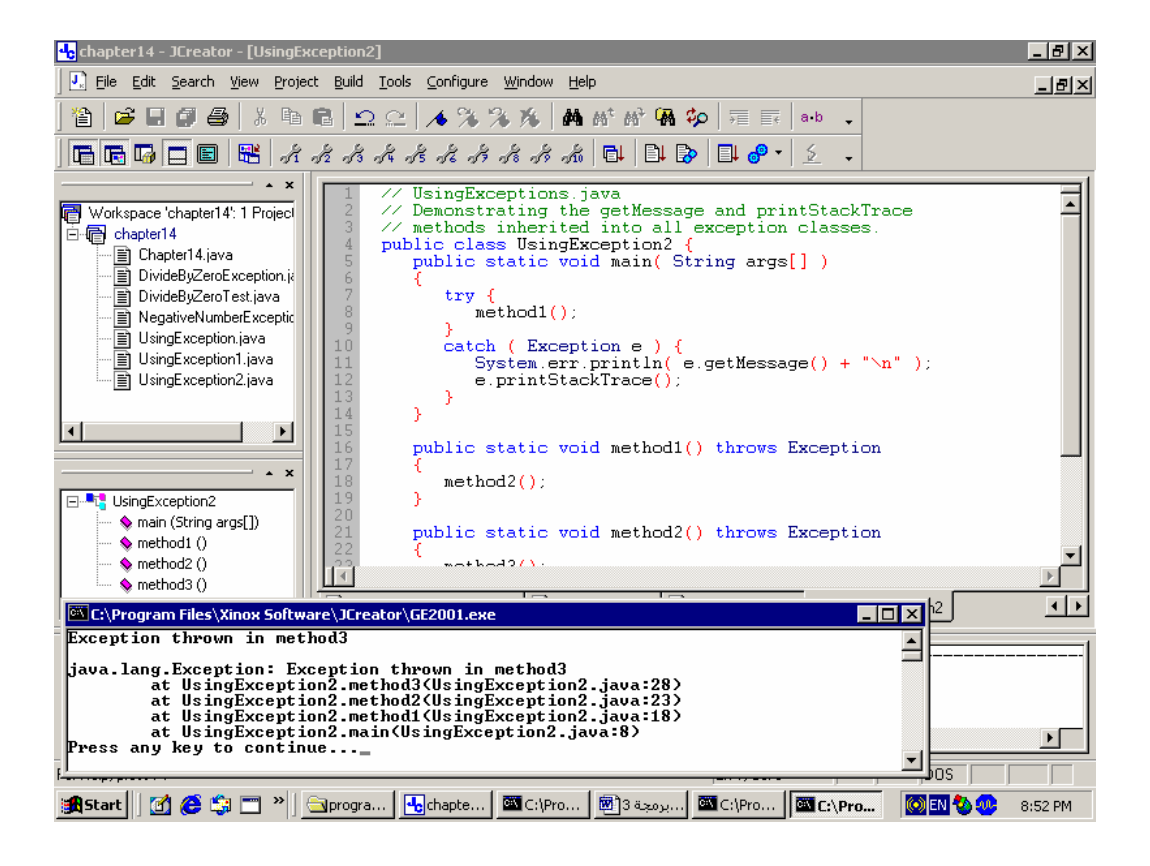
شكل (٢-٩)

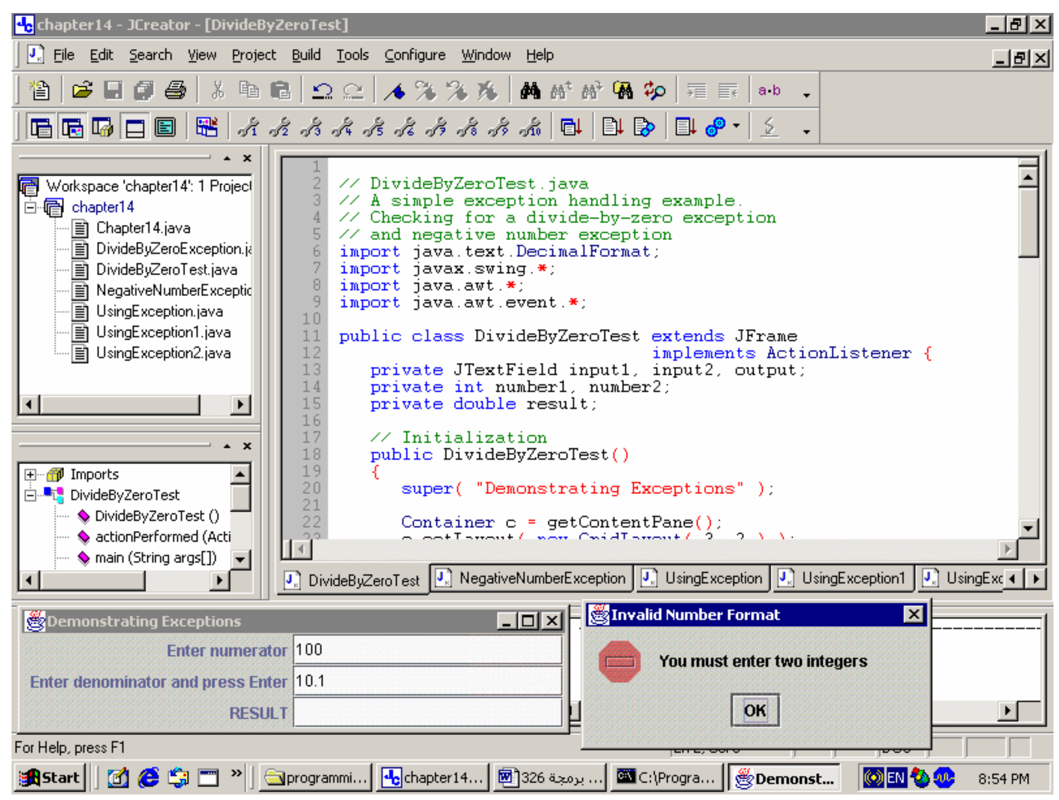


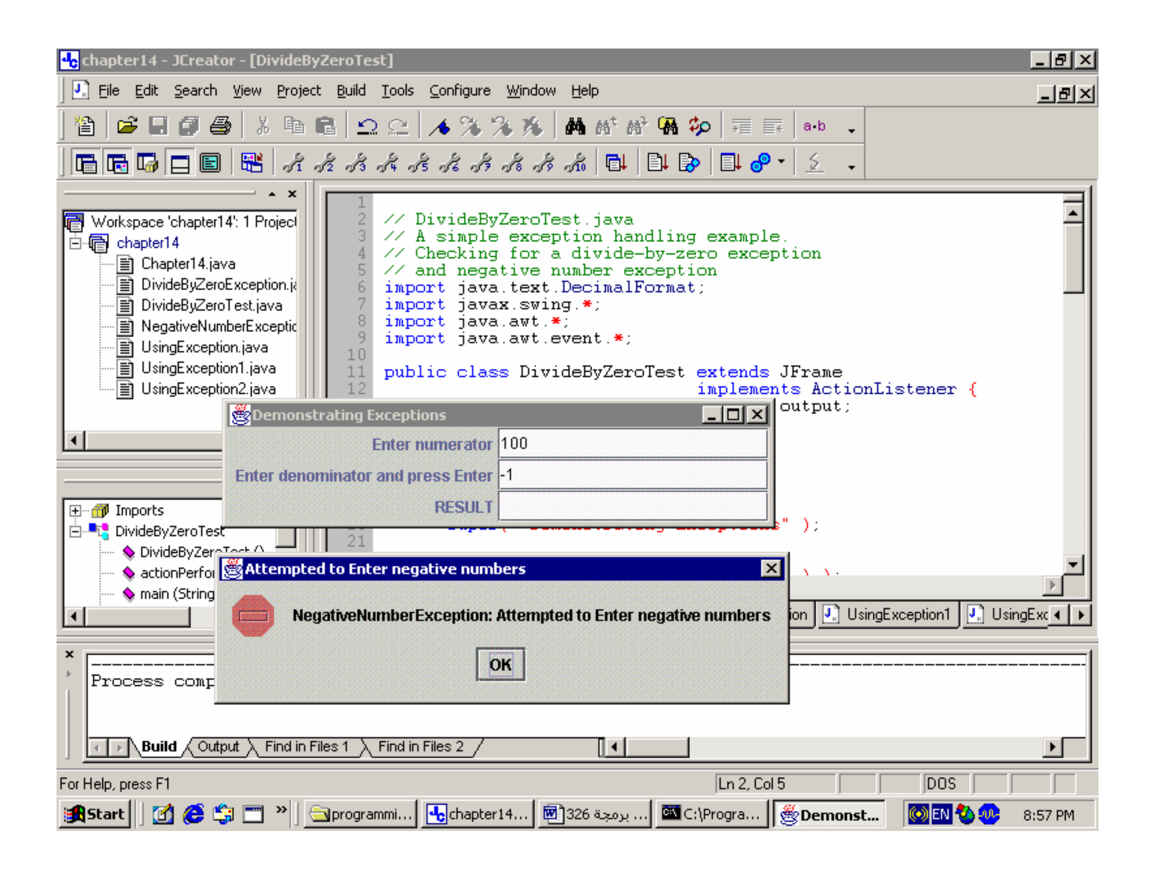
chapter14 - JCreator - [UsingException]
// UsingExceptions.java
// Demonstration of the try-catch-finally
// exception handling mechanism.
public class UsingException {
public static void main(String args[])
{
try {
throwException();
}
catch (Exception e)
{
System.err.println("Exception handled in main");
}
doesNotThrowException();
}
public static void throwException() throws Exception
{
// Throw an exception and immediately catch it.
try {
System.out.println("Method throwException");
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException
Press any key to continue...



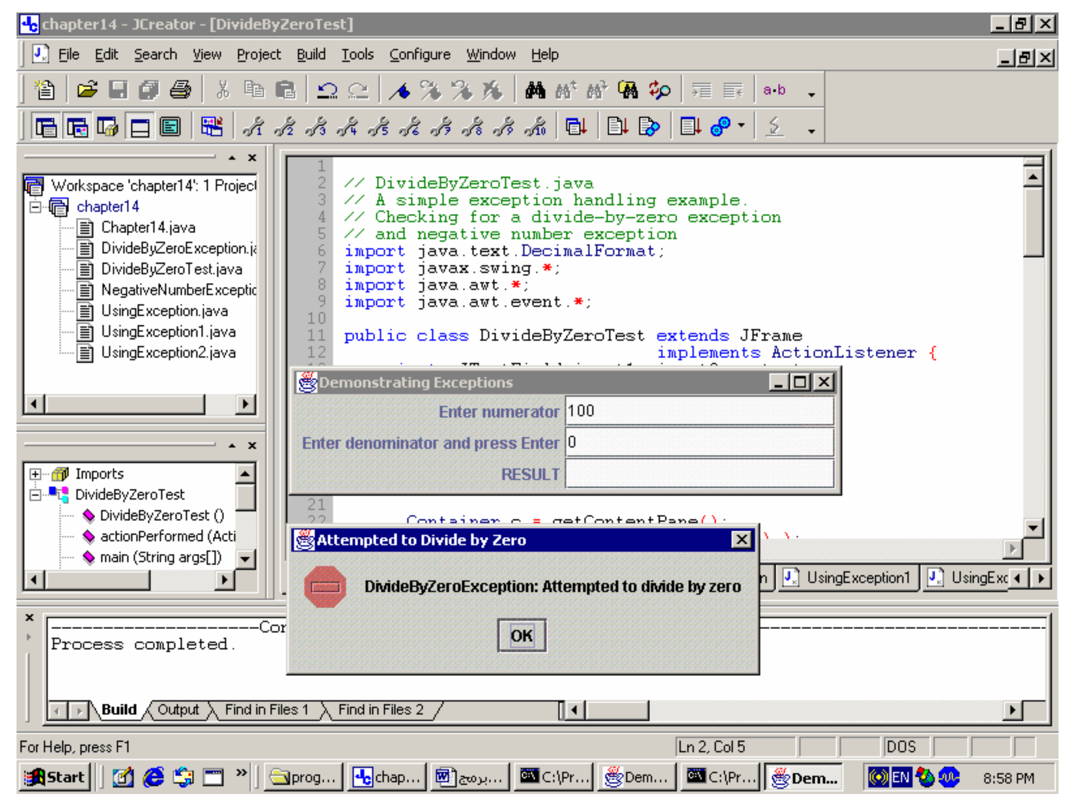
chapter14 - JCreator - [UsingException1]
// UsingExceptions.java
// Demonstration of stack unwinding.
public class UsingException1 {
public static void main(String args[])
{
try {
throwException();
}
catch (Exception e) {
System.err.println("Exception handled in main");
}
}
public static void throwException() throws Exception
{
// Throw an exception and catch it in main.
try {
System.out.println("Method throwException");
throw new Exception(); // generate exception
}
catch(RuntimeException e) { // nothing caught here
System.err.println("Exception handled in " +
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Method throwException
Finally is always executed
Exception handled in main
Press any key to continue...







chapter14 - JCreator - [DivideByZeroTest]
// DivideByZeroTest.java
// A simple exception handling example.
// Checking for a divide-by-zero exception
// and negative number exception
import java.text.DecimalFormat;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DivideByZeroTest extends JFrame
implements ActionListener {
Demonstrating Exceptions
Enter numerator 100
Enter denominator and press Enter -1
RESULT
Attempted to Enter negative numbers
NegativeNumberException: Attempted to Enter negative numbers
OK



chapter14 - JCreator - [DivideByZeroTest]
// DivideByZeroTest.java
// A simple exception handling example.
// Checking for a divide-by-zero exception
// and negative number exception
import java.text.DecimalFormat;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DivideByZeroTest extends JFrame
implements ActionListener {
Demonstrating Exceptions
Enter numerator 100
Enter denominator and press Enter 0
RESULT
Attempted to Divide by Zero
DivideByZeroException: Attempted to divide by zero
OK
Process completed.

DivideByZeroException.java
Created with JBuilder

```
// DivideByZeroException.java
// Definition of class DivideByZeroException.
// Used to throw an exception when a
// divide-by-zero is attempted.
 public class DivideByZeroException
          extends ArithmeticException {
  public DivideByZeroException()
  {
    super( "Attempted to divide by zero" );
  }

  public DivideByZeroException( String message )
  {
    super( message );
  }
```

NegativeNumberException.java
Created with JBuilder

```
// NegativeNumberException.java
// Definition of class NegativeNumberException.
// Used to throw an exception when a
// negative number is intered
 public class NegativeNumberException
          extends ArithmeticException {
  public NegativeNumberException()
  {
    super( "Attempted to Enter negative numbers" );
  }

}
```

DivideByZeroTest.java
Created with JBuilder

```
// DivideByZeroTest.java
// A simple exception handling example.
// Checking for a divide-by-zero exception
// and negative number exception
import java.text.DecimalFormat;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DivideByZeroTest extends JFrame
                    implements ActionListener {
  private JTextField input1, input2, output;
  private int number1, number2;
  private double result;

  // Initialization
  public DivideByZeroTest()
  {
    super( "Demonstrating Exceptions" );

    Container c = getContentPane();
    c.setLayout( new GridLayout( 3, 2 ) );

    c.add( new JLabel( "Enter numerator ",
               SwingConstants.RIGHT ) );
    input1 = new JTextField( 10 );
    c.add( input1 );

    c.add(
      new JLabel( "Enter denominator and press Enter ",
              SwingConstants.RIGHT ) );
    input2 = new JTextField( 10 );
    c.add( input2 );
    input2.addActionListener( this );

    c.add( new JLabel( "RESULT ", SwingConstants.RIGHT ) );
    output = new JTextField();
    c.add( output );

    setSize( 425, 100 );
    show();
  }
```

```
// Process GUI events
public void actionPerformed( ActionEvent e )
{
  DecimalFormat precision3 = new DecimalFormat( "0.000" );

  output.setText( "" ); // empty the output JTextField

  try {
    number1 = Integer.parseInt( input1.getText() );
    number2 = Integer.parseInt( input2.getText() );
                    if(number1 < 0 || number2 <0)
                     throw new NegativeNumberException();
    result = quotient( number1, number2 );
    output.setText( precision3.format( result ) );
  }
  catch ( NumberFormatException nfe ) {
    JOptionPane.showMessageDialog( this,
      "You must enter two integers",
      "Invalid Number Format",
      JOptionPane.ERROR_MESSAGE );
  }
  catch ( DivideByZeroException dbze ) {
    JOptionPane.showMessageDialog( this, dbze.toString(),
      "Attempted to Divide by Zero",
      JOptionPane.ERROR_MESSAGE );
  }
  catch ( NegativeNumberException nne ) {
    JOptionPane.showMessageDialog( this, nne.toString(),
      "Attempted to Enter negative numbers",
      JOptionPane.ERROR_MESSAGE );
  }
}

// Definition of method quotient. Used to demonstrate
// throwing an exception when a divide-by-zero error
// is encountered.
public double quotient( int numerator, int denominator )
  throws DivideByZeroException
{
  if ( denominator == 0 )
    throw new DivideByZeroException();

  return ( double ) numerator / denominator;
}

public static void main( String args[] )
{
  DivideByZeroTest app = new DivideByZeroTest();
```

```
     app.addWindowListener(
       new WindowAdapter() {
         public void windowClosing( WindowEvent e )
         {
           e.getWindow().dispose();
           System.exit( 0 );
         }
       }
     );
   }
 }
```

UsingException.java
Created with JBuilder

```
 // UsingExceptions.java
 // Demonstration of the try-catch-finally
 // exception handling mechanism.
 public class UsingException {
   public static void main( String args[] )
   {
     try {
       throwException();
     }
    catch ( Exception e )
    {
      System.err.println( "Exception handled in main" );
    }

    doesNotThrowException();
  }
  public static void throwException() throws Exception
  {
    // Throw an exception and immediately catch it.
    try {
      System.out.println( "Method throwException" );
      throw new Exception();  // generate exception
    }
    catch( Exception e )
    {
      System.err.println(
        "Exception handled in method throwException" );
      throw e;  // rethrow e for further processing

      // any code here would not be reached
    }
```

```
      finally {
        System.err.println(
          "Finally executed in throwException" );
      }

      // any code here would not be reached
    }
    public static void doesNotThrowException()
    {
      try {
        System.out.println( "Method doesNotThrowException" );
      }
      catch( Exception e )
      {
        System.err.println( e.toString() );
      }
      finally {
        System.err.println(
          "Finally executed in doesNotThrowException" );
      }
      System.out.println(
        "End of method doesNotThrowException" );
    }
  }
```

UsingException1.java

Created with JBuilder

```
// UsingExceptions.java
// Demonstration of stack unwinding.
public class UsingException1 {
  public static void main( String args[] )
  {
    try {
      throwException();
    }
    catch ( Exception e ) {
      System.err.println( "Exception handled in main" );
    }
  }

  public static void throwException() throws Exception
  {
    // Throw an exception and catch it in main.
    try {
      System.out.println( "Method throwException" );
      throw new Exception();      // generate exception
    }
    catch( RuntimeException e ) {  // nothing caught here
```

```
      System.err.println( "Exception handled in " +
                          "method throwException" );
    }
    finally {
      System.err.println( "Finally is always executed" );
    }
  }
}
```

UsingException2.java

Created with JBuilder

```
// UsingExceptions.java
// Demonstrating the getMessage and printStackTrace
// methods inherited into all exception classes.
public class UsingException2 {
  public static void main( String args[] )
  {
    try {
      method1();
    }
    catch ( Exception e ) {
      System.err.println( e.getMessage() + "\n" );
      e.printStackTrace();
    }
  }

  public static void method1() throws Exception
  {
    method2();
  }

  public static void method2() throws Exception
  {
    method3();
  }

  public static void method3() throws Exception
  {
    throw new Exception( "Exception thrown in method3" );
  }
}
```



برمجة ٣

معالجة الحدث

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات تصميم واجهات المستخدم، والقدرةعلى كتابة برامج لبناء مثل هذه الواجهات، واستخدام الأجزاء الرسومية المختلفة

الأهـداف:

بنهاية هذه الوحده، عليك ان تكون قادرا على كتابة برنامج جافا لتم بما يلي:

٧. فهم أساسيات تصميم الواجهات الرسومية.
٨. بناء واجهات التطبيق الرسومية
٩. انشاء ومعالجة الأجزاء الرسويةالمختلفة

مستوى الأاء المطلوب:

أ يصل المتدرب إلى إقان الجدارةبنسبة ١٠٠٪

الوسائل المساعدة

- وجود حاسب آي
- دفتر
- قلم

معالجة الحدث

Event Handling

مقدمة

إخال البيانات في البرامج التطبيقية التي تنفذ من خط الأوامر command line أو لوحة المراقبة console application دائما تكون تحت سيطرة البرنامج حيث يدخل المستخدم البيانات بترتيب معين ولكن البرامج التي تستخدمها يوميا في حاسبك لاتعمل بهذا الألوب وفيها يتم استخدام واجهة المستخدم الرسومية graphical user interface (GUI) ويكون التحكم لدى المستخدم حيث يمكنه استخدام كلاً من الفأرة ولوحة المفاتيح والتعامل مع واجهة المستخدم بأى ترتيب يريده ومثال ذلك يمكن للمستخدم إخال معلومات في مجال نصيText field أو اختيار من قائمة pull down menu أو الضغط على مفتاح click button أو إغلاق نافذةclose window والبرنامج يجب أن يتفاعل مع أوامر المستخدم بأي ترتيب تصل اليه والتعامل مع العديد من المدخلات الممكنة بترتيب عشوائي أمر صعب عند إلزام المستخدم ادخال البيانات بترتيب ثابت. في هذه الوحدة سوف نتعلم كيف نجعل برنامج جافا يستطيع أن يتفاعل مع أحداث واجهة المستخدم

الحدث Event الإنصات للحدث Event listeners ومصادر الحدث Event Sources

عندما يكتب مستخدم البرامج الرسومية مجموعة حروف أو يستخدم الفأرة في أي مكان داخل نافذة البرنامج يرسل مدير النافذة في الجافا java window manager إشعاراً للبرنامج أنه وقع حدث event وللعلم أن مدير النافذة يولد العديد من الأحداث events مثال ذلك عند تحريك الفأرة فترة قصيرة جداً على النافذة يولد حدث mouse event والكثير من البرامج لا تهتم بجميع الأحداث التي يولدها مدير النافذة ولذلك يجب على كل برنامج إيضاح أي الأحداث يحب استقبالها ويتم ذلك بإضافة كائنات استماع للحدث event listener وعلأوة على ذلك يوجد العديد من أنواع الأحداث مثال ذلك:

- أحداث لوحة المفاتيح Keyboard events
- أحداث حركة الفأرة mouse move events
- أحداث الضغط على الفأرة mouse click events
- أحداث غلق النافذة Window close events
- أحداث الضغط على زر Button click events

ولتجعل الاستماع للحدث أكثر تنظيماً تستخدم فصائل الاستماع للحدث event listener classes للاستماع للأنواع المختلفة من الأحداث.

ولتثبيت مستمع للحدث event listener يجب أن تعرف مصدر الحدث event source ومصدر الحدث هو مكونة واجهة المستخدم user interface component التي تولد حدثاً معيناً مثال ذلك:

- الزر button هو مصدر الحدث لإحداث الضغط على الزر Button click events
- القائمة menu هي مصدر حدث اختيار القائمة menu selection event
- شريط الزلاقة scrollbar هو مصدر حدث ضبط شريط الزلاقة scrollbar adjustment event

ويجب أن تخبر مصدر الحدث event source أى مستمع للأحداث تريد تثبيته.

مثال:

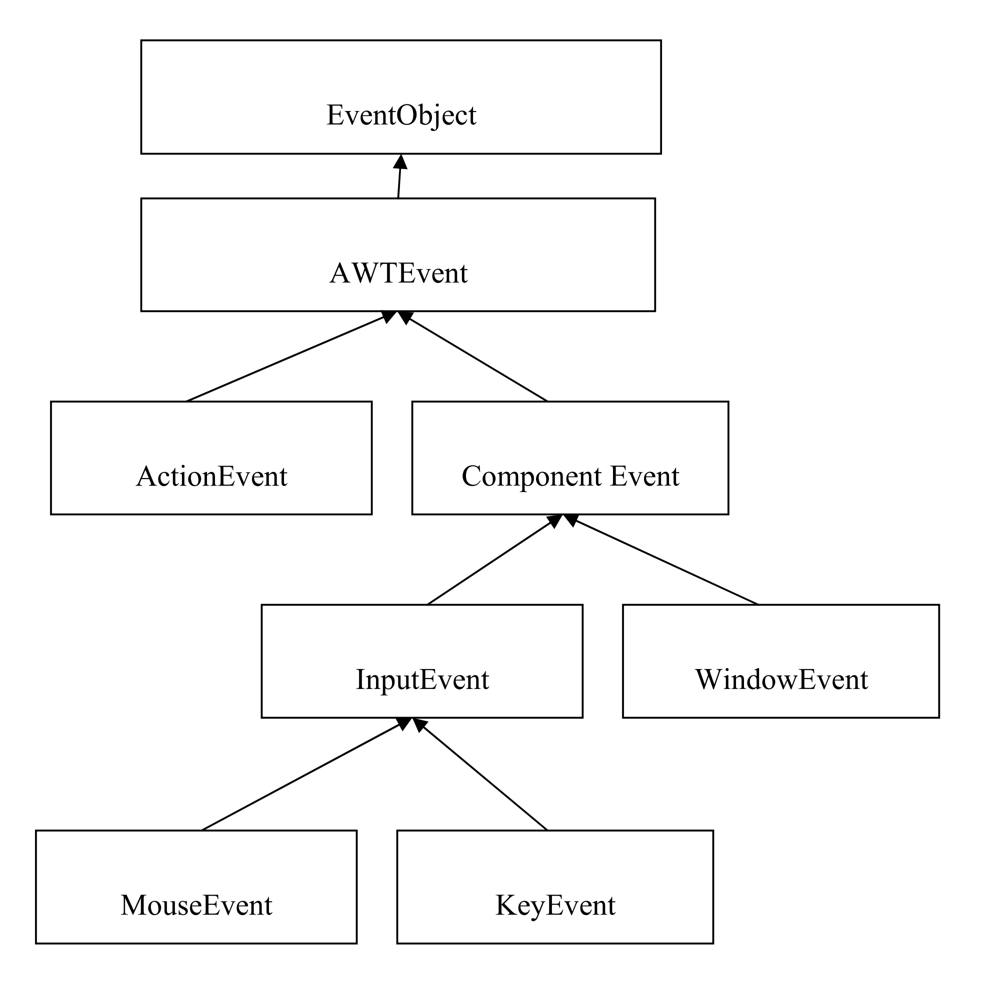
في هذا المثال سوف نستمع لحدث الضغط على الفأرة في برمجيات an applet ويوجد ثلاث فصائل تشمل هذا الحدث:

١ - **فصيلة الحدث The event class** في حالة الضغط على الفأرة توجد الفصيلة MouseEvent حيث إن الكائن المنشىء من هذه الفصيلة يخبر عن موضع مؤشر الفأرة (x و y) وما هو زر الفأرة الذى ضغطه المستخدم

٢ - **فصيلة المستمع The listener class** توجد الفصيلة MouseListener التي تستخدم لتنفيذ واجهة الاستماع للفأرة ولها العديد من الطرق التي تستدعى عند الضغط على زر الفأرة أو عند الرفع من على زر الفأرة ولكل هذه الطرق عوامل للفصيلة MouseEvent

٣ - **مصدر الحدث The event source** هذه هي المكونة component التي تولد حدث الفأرة وتدير الاستماع وفي هذا المثال يكون مصدر الحدث هو البرمجيات the applet التي ربما يضغط المستخدم الفأرة على مساحة سطحها وهنا يجب أن نخبرها ما هو مستمع الفأرة يجب اشعارها به عند وقوع حدث الفأرة.

كل فصائل الحدث event classes هي فصائل فرعية من الفصيلة EventObject ويبين شكل (٣ -١) مخطط الوراثة لمعظم فصائل الحدث. الفصيلة EventObject تمتلك الطريقة المهمة getSource التي تسترجع الكائن الذى يولد الحدث والفصائل الفرعية تمتلك الطرق الخاصة بها التي تصف الحدث أكثر ومثال ذلك الفصيلة MouseEvent تمتلك الطريقة getX و الطريقة getY التي تخبر عن موقع الفأرة أثناء توليد الحدث.



شكل (٣ -١)

الجزء الأكثر تعقيداً في معالجة الحدث في لغة الجافا هو الاقتراب من المستمع ومستمع الفأرة يجب أن ينفذ الخمس طرق التالية من الرابط MouseListener interface

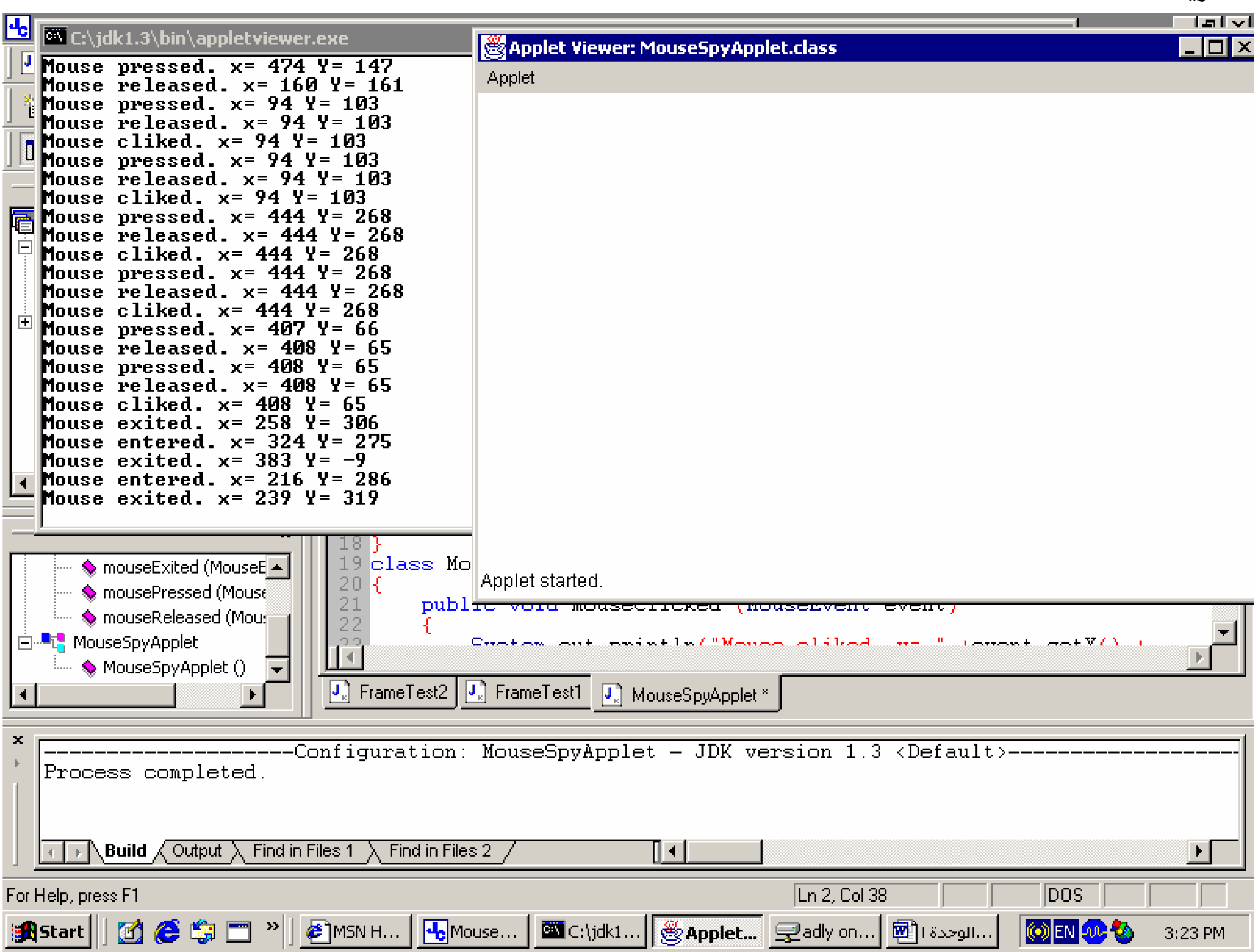
```
public interface Mouselistener
{
        void mouseClicked (MouseEvent event)
                // called when the mouse has been clicked on a component
        void mouseEntered (MouseEvent event)
                // called when the mouse enters a component
        void mouseExited (MouseEvent event)
                // called when the mouse exits a component
        void mousePressed (MouseEvent event)
                // called when the mouse button has been presseded on a component
        void mouseReleaseded (MouseEvent event)
                // called when the mouse button has been releaseded on a component
}
```

والآن نريد أن نراقب أحداث الفأرة ونظهرها كما تحدث ولهذا الغرض تم إنشاء الفصيلة MouseSpy لتنفذ طرق المستمع في الفصيلة Mouselistener بحيث نظهر سبب الحدث وموضع الفأرة (x و Y) ويبين شكل (٣ -٢) برنامجاً لتنفيذ هذه الفصيلة والطرق جميعها في الأسطر من ١٧ إلى ٤٤ والآن نريد أن نثبت المستمع ولذلك تم استدعاء الطريقة addMouseListener لمصدر الحدث وهو applet وتم إنشاء كائن من الفصيلة MouseSpy وتمريره كمعامل للطريقة addMouseListener كما هو مبين في البرنامج في الأسطر ١٣ و ١٤

```
1.  //listen to mouse evnts in an applet
2.  import java.awt.*;
3.  import java.applet.*;
4.  import java.awt.event.MouseListener;
5.  import java.awt.event.MouseEvent;
6.  public class MouseSpyApplet extends Applet
7.  {
8.  public MouseSpyApplet()
9.    {
10.     MouseSpy listener = new MouseSpy();
11.     addMouseListener(listener);
12.   }
13. }
14. class MouseSpy implements MouseListener
15. {
16. public void mouseClicked (MouseEvent event)
17.   {
18.        System.out.println("Mouse cliked. x= " +event.getX() +
19.        " Y= " +event.getY());
20.   }
21. public void mouseEntered (MouseEvent event)
22.   {
23.      System.out.println("Mouse entered. x= " +event.getX() +
24.      " Y= " +event.getY());
25.   }
26. public void mouseExited (MouseEvent event)
27.   {
28.       System.out.println("Mouse exited. x= " +event.getX() +
29.       " Y= " +event.getY());
30.   }
31. public void mousePressed (MouseEvent event)
32.   {
33.      System.out.println("Mouse pressed. x= " +event.getX() +
34.      " Y= " +event.getY());
35.   }
36. public void mouseReleased (MouseEvent event)
37.   {
38.      System.out.println("Mouse released. x= " +event.getX() +
39.      " Y= " +event.getY());
40.   }
41. }
```

شكل (٣-٢)

ولمتابعة هذا البرنامج والاستفادة منه يجب تنفيذه ويكون ناتج هذا البرنامج كما هو مبين في شكل (٣- ٣) انه كلما أحست البرمجيات the applet بحدث الفأرة تستدعى الطريقة المناسبة لكائن المستمع كمثال لذلك في السطر الأول من شكل (٣ -٣) نجد أن المستخدم ضغط على زر الفأرة في الموقع x=474 و y=147 فتم استدعاء الطريقة mousePressed والسطر الثاني يبين أن المستخدم رفع يده عن زر الفأرة في الموقع x=168 و y=161 فتم استدعاء الطريقة mouseReleased والسطر الثالث والرابع والخامس يبين أن المستخدم ضغط ورفع بسرعة عند الموقع x=94 و y=103 ولذلك تم استدعاء الطرق mousePressed و mouseReleased و mouseClicked والسطر قبل الأخير من الشكل يبين أن الفأرة دخلت سطح البرمجيات applet في الموقع x=216 و y=286 ولذلك تم استدعاء الطريقة mouseEntered وخرجت الفأرة من سطح البرمجيات في الموقع x=239 و y=319 ولذلك تم استدعاء الطريقة mouseExited.



شكل(٣ -٣)

مهايأة الحدث Event Adapter

في الجزء السابق رأينا كيفية تثبيت مستمع listener لمصدر حدث الفأرة وكيفية استدعاء طرق المستمع listener methods عند وقوع الحدث . وعادة البرنامج لايهتم بكل إشعارات المستمع ومثال ذلك أن البرنامج يريد الاستماع فقط للضغط السريع على زر الفأرة mouse click ولايهتم أن الضغط السريع يتكون من حدث الضغط على زر الفأرة mouse pressed و الرفع من على زر الفأرة mouse released وبالطبع يمكن للبرنامج إعطاء المستمع الذى يعرف جميع الطرق methods التي لايهتم بها البرنامج أن لاتفعل شيئاً كالتالي:

```
1.  class MouseClickListener implements MouseListener
2.  {
3.  public void mouseClicked (MouseEvent event)
4.    {
5.      // mouse click action here
6.    }

7.  public void mouseEntered (MouseEvent event)
8.     {
9.         // do nothing
10.  }
11. public void mouseExited (MouseEvent event)
12.  {
13.      // do nothing
14.  }
15. public void mousePressed (MouseEvent event)
16.  {
17.     // do nothing
18.
19.  }
20. public void mouseReleased (MouseEvent event)
21.  {
22.    // do nothing
23.
24.  }
25. }
```

ويوجد في لغة الجافا فصيلة مهيء adapter class الذى ينفذ الرابط MouseListener interface حيث إن جميع الطرق methods لاتفعل شيئاً do nothing كالتالي

```
1.  class MouseAdapter implements MouseListener
2.  // This class is defined in the java.awt.event package
3.  {
4.  public void mouseClicked (MouseEvent event)
5.    {
6.                // do nothing
7.    }
8.  public void mouseEntered (MouseEvent event)
9.     {
```

```
10.         // do nothing
11.  }
12. public void mouseExited (MouseEvent event)
13.  {
14.       // do nothing
15.  }
16. public void mousePressed (MouseEvent event)
17.  {
18.      // do nothing
19.  }
20. public void mouseReleased (MouseEvent event)
21.  {
22.     // do nothing
23.  }
24. }
```

وبذلك يمكنك تعريف فصيلة مستمع للضغط على الفأرة ترث .الفصيلة MouseAddapter وتكتب فقط الطرق التي تهتم بها فقط كالتإلي

```
class MouseClickListener extends MouseAdapter
{
         public void mouseClicked (MouseEvent event)
          {
                  // mouse click action here
          }
```

العديد من واجهات المستمع listener interface مثل واجهة مستمع الفعل ActionListener تمتلك طريقة واحدة فقط وفي هذه الحالة لايوجد مهيء مطابق حيث يمكن تنفيذ الواجهة implement interface بنفس سهولة وراثة (امتداد) المهيء extend the adapter ولكن الحزمة java.awt.event تحتوي على فصيلة مهيءadapter class لجميع واجهات مستمع الحدث التي تمتلك على الأقل طريقتين two methods ومثال ذلك مستمع النافذة WindowListener واجهة تحتوي على ٧ طرق ولذلك يوجد مهيء مطابق ينفذ الطرق السبعة بحيث لاتفعل شيئاً وسوف نستخدمه في هذه الوحدة لاحقاَ

تنفيذ المستمع كفصيلة داخلية Implementing Listener as inner class

في المثال السابق مستمع الفأرة كان يطبع كل أحداث الفأرة باستخدام System.out والأن نفترض أنك تريد شيئاً أكثر جاذبية يحدث عندما يضغط المستمع على الفأرة سريعاً وهنا في شكل (٣ -٤) برنامج يرسم قطع ناقص ellipse على الشاشة ويحرك القطع الناقص إلى موضع ضغط زر الفأرة.

إذا أردنا رسم قطع ناقص فقط يكون البرنامج بالشكل الآتي:

```
import java.awt.*;
```

```
import java.applet.*;
import java.awt.event.*;
import java.awt.geom.*;

public class EggApplet extends Applet
{
        public EggApplet()
        {
                egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);
        }
        public void paint(Graphics g)
        {
                Graphics2D g2 = (Graphics2D)g;
                g2.draw(egg);
        }
        private Ellipse2D.Double egg;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HIGHT = 50;
```

والآن دعنا نضيف مستمع للفأرة ونحأول تحريك القطع الناقص إلى موضع الفأرة ويصبح البرنامج كالتالي:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.geom.*;

public class EggApplet extends Applet
{
        public EggApplet()
        {
                egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);

                // add mouse click listener
                MouseClickListener listener = new MouseClickListener();
                addMouseListener(listener);
        }
        public void paint(Graphics g)
        {
                Graphics2D g2 = (Graphics2D)g;
                g2.draw(egg);
        }
        private Ellipse2D.Double egg;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HIGHT = 50;

        class MouseClickListener extends MouseAdapter
        {
                public void mouseClicked(MouseEvent event)
```

```
int mouseX=event.getX();
int mouseY=event.getY();

// now move the ellipse to (mouseX, MouseY)

egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
EGG_WIDTH, EGG_HIGHT);
repaint();
```

عند ترجمة البرنامج نجد أنه يعطي الأخطاء التالية وذلك لأن الفصيلة MouseClickListener ليس لها الحق في استخدام متغيرات الكائن الخاصة بالفصيلة EggApplet.

```
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable EGG_WIDTH
location: class MouseClickListener
                              egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                                   ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable EGG_HIGHT
location: class MouseClickListener
                              egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                                                ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:53: cannot resolve symbol
symbol  : variable EGG_WIDTH
location: class MouseClickListener
                              EGG_WIDTH, EGG_HIGHT);
                ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:53: cannot resolve symbol
symbol  : variable EGG_HIGHT
location: class MouseClickListener
                              EGG_WIDTH, EGG_HIGHT);
                              ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable egg
location: class MouseClickListener
                              egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:54: cannot resolve symbol
symbol  : method repaint  ()
location: class MouseClickListener
                              repaint();
                ^
6 errors
Process completed.
```

وهذه الحالة قياسية لمستمع الحدث event listener لأنه عادة تحتاج طرق الحدث للوصول إلى متغيرات في فصيلة آخرى وذلك يمكن علاجه باستخدام المستمع كفصيلة داخلية inner class

والفصيلة الداخلية تعرف داخل فصيلة آخرى وتتعامل كأنها فصيلة عادية حيث يمكن إنشاء كائنات والتفاعل مع الطرق بالأساليب المعتادة ولكن هناك استثناء حيث إن الطرق في الفصيلة الداخلية يسمح لها باستخدام متغيرات الكائن instance variables وطرق الفصيلة الخارجية وتكون الصيغة العامة في لغة الجافا كالتالي:

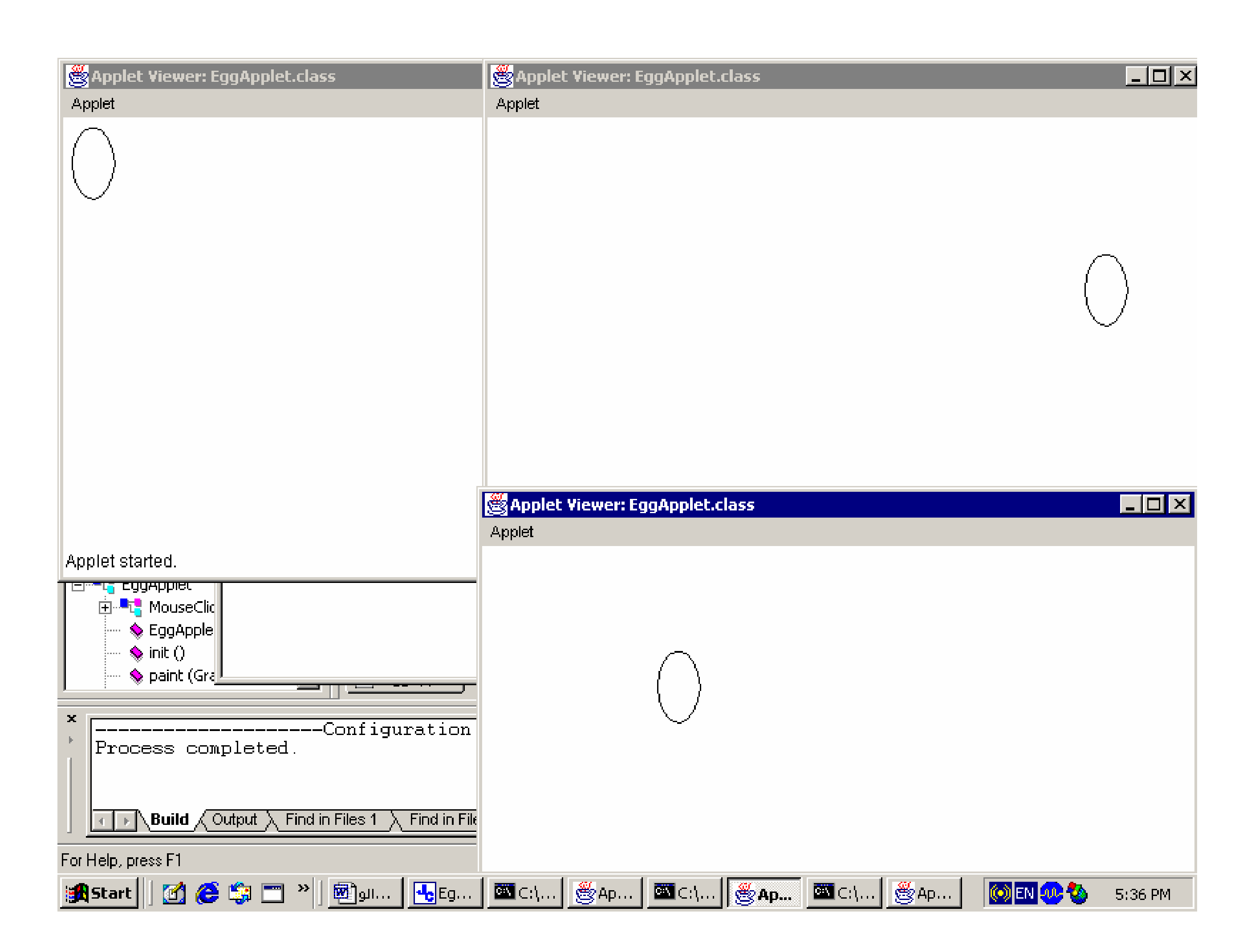
```
Class   OuterClassName
{
        .........
        accessSpecifier          class    InnerClassName
        {
                methods
                variables
        }
```

شكل (٣ -٤) يمثل البرنامج حيث تم ترجمته بشكل صحيح وتشغيله كما هو مبين في شكل (٣ -٥)

```
1.  // using inner class
2.  import java.awt.*;
3.  import java.applet.*;
4.  import java.awt.event.*;
5.  import java.awt.geom.*;
6.  public class EggApplet extends Applet
7.  {
8.  public EggApplet()
9.  {
10. egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);
11. // add mouse click listener
12. MouseClickListener listener = new MouseClickListener();
13. addMouseListener(listener);
14. }
15. public void paint(Graphics g)
16. {
17. Graphics2D g2 = (Graphics2D)g;
18. g2.draw(egg);
19. }
20. private Ellipse2D.Double egg;
21. private static final double EGG_WIDTH = 30;
22. private static final double EGG_HIGHT = 50;
23. // inner class definition
24.
25. private class MouseClickListener extends MouseAdapter
26. {
```

```
27. public void mouseClicked(MouseEvent event)
28. {
29. int mouseX=event.getX();
30. int mouseY=event.getY();
31. // now move the ellipse to (mouseX, MouseY)
32. egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
33. EGG_WIDTH, EGG_HIGHT);
34. repaint();
35. }
36. }
37. }
```

شكل(٣ -٤)



شكل(٣ -٥)

نوافذ الإطار Frame Windows

جميع البرامج الرسومية التي كتبت سابقا استخدمت البرمجيات applets ويمكنك الآن تعلم كتابة برامج رسومية من خلال تطبيقات الجافا java applications وكل برنامج رسومي يستخدم نافذة إطار frame window أو أكثر ونافذة الإطار لها شريط عنوان title bar و حد border

لكي تظهر الإطار نستخدم الفصيلة JFrame من الحزمة javax.swing ويجب تحديد مقاس الإطار باستدعاء الطريقة setSize ولتحديد عنوان الإطار نستخدم الطريقة setTitle والطريقة show تجعل مدير النافذة window manager يظهر الإطار ويبين شكل (٣ -٦) برنامج بسيط لإظهار إطار نافذة بدون أي شيء داخله وكما تعلمنا سابقاً أن تطبيقات الجافا لابد أن تحتوي على الطريقة الرئيسة main وفي السطر ٤ يتم إنشاء الكائن frame من الفصيلة EmptyFrame التي هي امتداد للفصيلةJFrame وفي منشيء الفصيلة EmptyFrame يتم تحديد حجم الإطار وذلك في الأسطر من ٩ إلى ١٧ وفي الطريقة main أيضاً يتم تحديد عنوان الإطار في السطر ٥ و إظهار الإطار في السطر ٦

```
1.  import javax.swing.JFrame;
2.  public class FrameTest1
3.  { public static void main (String[] args)
4.      { EmptyFrame frame = new EmptyFrame();
5.        frame.setTitle("frameTest");
6.        frame.show();
7.      }
8.  }
9.  class EmptyFrame extends JFrame
10. {
11. public EmptyFrame()
12.   {
13.     final int DEFAULT_FRAME_WIDTH =300;
14.     final int DEFAULT_FRAME_HIGHT =300;
15.     setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HIGHT);
16.   }
17. }
```

شكل (٣ -٦)

عند تنفيذ البرنامج يتم إظهار الإطار وينتهي تنفيذ الطريقة main ولكن يظل البرنامج يعمل ويبقى الإطار ظاهر على الشاشة ويمكن تحريكه وتغيير حجمه وهذا هو الفرق الأساسي بين البرامج التي تعمل من لوحة المراقبة console programs والبرامج الرسومية graphical programs

بمجرد إظهار نافذة الإطار يبدأ البرنامج خيطاً جديداً new thread لتنفيذ إظهار واجهة المستخدم الرسومية وعند انتهاء الطريقة main ينتهي خيط الطريقة main thread ولكن برنامج الجافا لاينتهي لأن خيط

واجهة المستخدم مازال يعمل وهذه مشكلة حيث إنه عند إغلاق نافذة الإطار بالضغط على أيقونة الغلق من شريط العنوان يظل البرنامج يشتغل ولايعمل شيئاً ولإنهاء البرنامج يجب استخدام الجملة التالية

System.exit(0)

والمشكلة أين تضع هذه الجملة. لايمكن وضعها في نهاية الطريقة main كالتالي

```
public class FrameTest1
        public static void main (String[] args)
    { EmptyFrame frame = new EmptyFrame();
      frame.setTitle("frameTest");
      frame.show();
      System.exit(0);    // Error
```

بهذا الوضع يظهر البرنامج النافذة للحظة وجيزة وينتهي فوراً وبدلاً عن ذلك نريد إنهاء البرنامج عندما يضغط المستخدم على أيقونة الغلق في شريط العنوان ولكن لانعرف متى يحدث ذلك لأن المستخدم هو المتحكم في البرنامج ويمكن عمل العديد من الأشياء بأي ترتيب والحل هو تثبيت معالج للحدث الذي يستدعى عندما يضغط المستخدم على أيقونة الغلق واستجابة لهذا الحدث ننهي البرنامج

ولتكتشف متى يغلق المستخدم النافذة يجب الاستماع لإحداث النافذة window events ويمكن حدوث سبعة أحداث للنافذة في برنامج الجافا كالتالي

١. النافذة فتحت الآن للمرة الأولى
٢. تم إغلاق النافذة نتيجة للطريقة dispose
٣. تم تنشيط النافذة بسبب الضغط داخلها
٤. تم إيقاف نشاط النافذة نتيجة للدخول في نافذة آخرى
٥. تم تحويل النافذة إلى أيقونة نتيجة الضغط على أيقونة التصغير في شريط العنوان
٦. تم استرجاع النافذة (من حالة أيقونة) نتيجة للضغط على أيقونة النافذة المصغرة
٧. تم غلق النافذة نتيجة الضغط على أيقونة الغلق في شريط العنوان

وللإستماع لهذه الأحداث يجب إضافة كائن مستمع النافذة window listener إلى الإطار وكائن مستمع النافذة ينفذ الرابط WindowListener الذي يمتلك سبع طرق كالتالي:

```
Public interface WindowListener

        void windowOpend(windowEvent e);
        void windowClosed(windowEvent e);
        void windowActivated(windowEvent e);
        void windowDeactivated(windowEvent e);
        void windowIconified(windowEvent e);
        void windowDeiconfied(windowEvent e);
```

void windowClosing(windowEvent e);

بالطبع البرنامج البسيط لايهتم بالأحداث الستة الأولى لأن الفصيلة العليا JFrame تستمع إليهم وتعرف كيف تتعامل معهم ولكن هناك برامج رسومية متخصصة تحتاج أن تعرف عن بعض هذه الأحداث مثال ذلك البرنامج الذي يظهر الرسوم المتحركة يريد أن يوقف الحركة عندما تصغر النافذة ويبدأ الحركة عند استعادة النافذة. ربما أن تتعجب لماذا لايستدعى كائن الفصيلة JFrame ببساطة الطريقة System.exit عندما يتم إغلاقه وذلك لسبب بسيط وهو أن البرنامج يمكن أن يمتلك العديد من نوافذ الإطار وليست فكرة طيبة أن ينتهي البرنامج كله إذا أغلق المستخدم إحدى نوافذه ولذلك يجب أن نعلم نافذة معينة أن تنتهي عندما يغلقها المستخدم ويتحقق ذلك بتثبيت كائن مستمع نافذة فيه الطريقة windowClosing لتنهي البرنامج ويمكن استخدام المهيء WindowAdapter الذى ينفذ طرق الرابط WindowListener بحيث لاتعمل شيئاً nothing do وهنا يتم إنشاء الفصيلة WindowCloser التي ترث الفصيلة WindowAdapterكالتإلى

```
class WindowCloser extends WindowAdapter

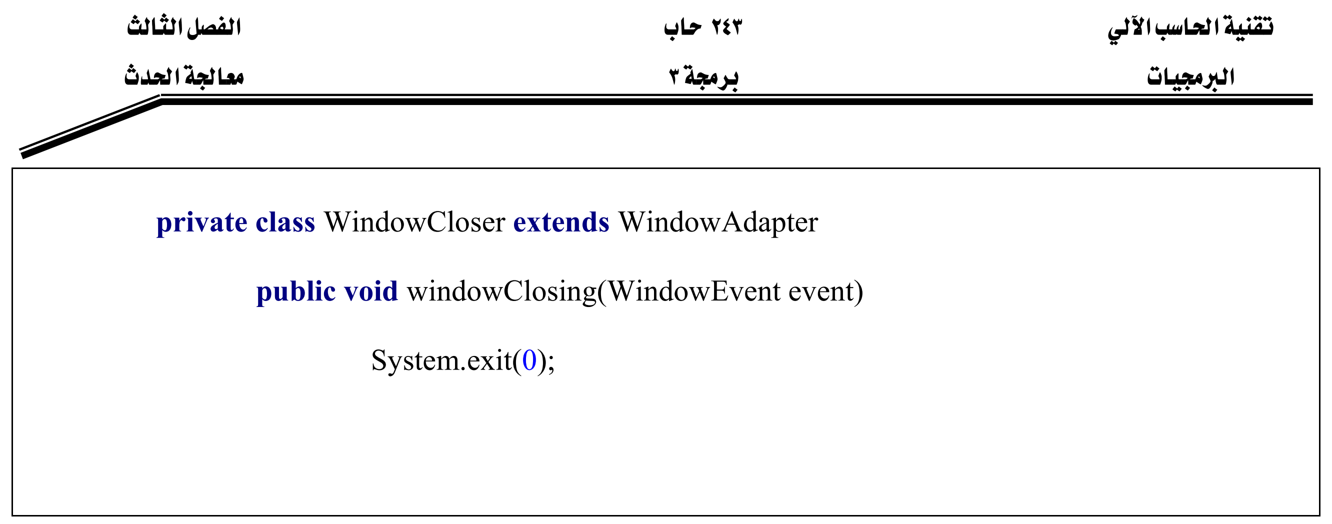
        public void windowClosing(WindowEvent event)

                System.exit(0);
```

وفي النهاية نحتاج أن نضيف كائن من الفصيلة WindowCloser كمستمع للنافذة للإطار باستدعاء الطريقة addWindowListener ويصبح البرنامج كما هو مبين في شكل (٣ -٧)

```
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FrameTest2
{           public static void main (String[] args)
            { EmptyFrame frame = new EmptyFrame();
                        frame.setTitle("Close me!");
                        frame.show();
            }
}
class EmptyFrame extends JFrame
{public EmptyFrame()
            {           final int DEFAULT_FRAME_WIDTH =300;
                        final int DEFAULT_FRAME_HIGHT =300;
                        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HIGHT);

                         WindowCloser listener = new WindowCloser();
                         addWindowListener(listener1);
```



```
private class WindowCloser extends WindowAdapter

    public void windowClosing(WindowEvent event)

        System.exit(0);
```

شكل(٣ -٧)

والآن يمكن إنهاء التطبيق بصورة صحيحة عندما يغلق المستخدم نافذة الإطار

شكل(٣ -٩)

المملكة العربية السعودية
المؤسسة العامة للتعليم الفني والتدريب المهني
الإدارة العامة لتصميم وتطوير المناهج



برمجة ٣

واجهات المستخدم الرسومية

واجهات المستخدم الرسومية

٤

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات تصميم واجهات المستخدم، والقدرة على كتابة برامج لبناء مثل هذه الواجهات، واستخدام الأجزاء الرسومية المختلفه.

الأهـداف:

بنهاية هذه الوحده، عليك أن تكون قادرا على:

١. فهم أساسيات تطبيقات الواجهات الرسومية.
٢. بناء واجهات التطبيق الرسومية
٣. إنشاء ومعالجة الأجزاء الرسومية المختلفة.
٤. كتابة برنامج جافا لدعم المفاهيم السابقة.

مستوى الاداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعده:

- وجود حاسب آلي يحتوي على بيئة متكاملة لكتابة برامج بلغة جافا
- دفتر
- قلم

واجهات المستخدم الرسومية

مقدمة :

يعتبر بناء واجهات المستخدم الرسومية من الأجزاء المهمة في البرنامج، حيث إن هذه الواجهات تعطي البرنامج شكلا معينا وشعورا معينا لدى المستخدم، حيث إن استخدام مفاهيم وأجزاء موحدة في بناء الواجهات للعديد من البرامج المختلفة يعطي المستخدم قدراً كبيراً من الراحة أثناء استخدام البرامج ويقلل من الوقت اللازم لتعلمها. هناك العديد من الواجهات المستخدمة التي قمت باستخدامها أثناء استعمالك للحاسب فشاشات نظام الويندوز والمستكشف وغيرها تستخدم واجهات مستخدم رسومية.

تتكون واجهة المستخدم الرسومية من العديد من المكونات components وهي عباره عن كائنات objects يستطيع المستخدم التعامل معها بواسطة الفأرة، لوحة المفاتيح وغيرها من الوسائل، والجدول التالي يوضح بعض المكونات الرسومية لواجهة التطبيق:

اسم الجزء	الوصف
JLabel	مكان يوضع فيه نص أو صوره لا يمكن تغييره أو الكتابة عليه
JTextField	مكان يمكن أن يستقبل مدخلات من المستخدم وطباعة الناتج عليه.
JButton	مكان يقوم بإطلاق حدث ما عند الضغط عليه
JCheckBox	هو شكل رسومي يمكن أن يكون مختارا أو غير مختار
JcomboBox	هو عبارة عن قائمة من العناصر، يمكن للمستخدم الاختيار منها بالضغط على العنصر من القائمة.
JList	مكان يمكن لقائمة من العناصر أن تظهر فيه ليقوم المستخدم باختيار عنصر ما بالضغط عليه مرة بالفأرة.
JPanel	عباره عن حأوي لمجموعة العناصر الرسومية.

مراجعه للحزمة swing

إن جميع الأجزاء الموضحة في الجدول أعلاه محتواه داخل الحزمة الرسومية المسماة javax.swing وقد أصبحت هذه الأجزاء الرسومية أساسية في لغة جافا في الإصدار Java 2 platform النسخه 1.2. كما أن معظم أجزاء الحزمة swing تم كتابتها،و معالجتها وعرضها كليا بلغة جافا، لذلك فهي تسمى Pure Java Components.

إن الأجزاء الرسومية الأصليه في جافا والموجودة في الحزمة AWT مرتبطة مباشرة مع الإمكانيات الرسومية للجهاز المستخدمة فيه، لذلك فإن أجزاء GUI ستظهر بشكل متباين على الاجهزه المختلفه وذلك لاختلاف platform على كل منها؛ عند كتابة برنامج يقوم برسم زر button على نظام التشغيل ويندوز فإن هذا الزر سيكون له نفس شكل الزر في نظام ويندوز، وعند رسمه على نظام Apple Macintosh سيكون له شكل الزر في نظام Apple Macintosh. لذلك وحيث إن الأجزاء الرسومية في الحزمة swing بلغة جافا أصلا، فإن الواجهات الرسومية التي تستخدم هذه الحزمة ستحافظ على شكلها ومظهرها حتى لو اختلف نظام التشغيل من جهاز إلى آخر. وسنستعرض الآن أهم عناصر ومكونات الحزمة swing

العنصر الرسومي JLabel

يوفر العنصر الرسومي JLabel تعليمات نصية أو معلومات في واجهة المستخدم الرسومية GUI. يعرف هذا العنصر الرسومي من خلال الفصيلة JLabel ويظهر سطر نصي واحد، أو صورة أو كلاهما، للقراءة فقط ولا يمكن التعديل عليه. المثال التالي سيعرفك على العديد من الطرق الخاصة بالعنصر الرسومي JLabel:

```
1
2     // Demonstrating the JLabel class.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
7
8     // Java extension packages
9     import javax.swing.*;
10
11    public class LabelTest extends JFrame {
12       private JLabel label1, label2, label3;
13
14       // set up GUI
15       public LabelTest()
16       {
17          super( "Testing JLabel" );
18
19          // get content pane and set its layout
20          Container container = getContentPane();
21          container.setLayout( new FlowLayout() );
22
23          // JLabel constructor with a string argument
```

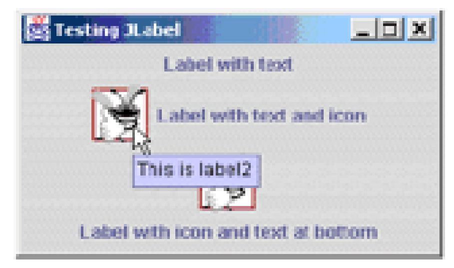
```
24          label1 = new JLabel( "Label with text" );
25          label1.setToolTipText( "This is label1" );

26          container.add( label1 );
27
28          // JLabel constructor with string, Icon and
29          // alignment arguments

30          Icon bug = new ImageIcon( "bug1.gif" );
31          label2 = new JLabel( "Label with text and icon",
32             bug, SwingConstants.LEFT );

33          label2.setToolTipText( "This is label2" );
34          container.add( label2 );
35
36          // JLabel constructor no arguments
37          label3 = new JLabel();
38          label3.setText( "Label with icon and text at bottom" );
39          label3.setIcon( bug );
40          abel3.setHorizontalTextPosition(SwingConstants.CENTER );
41          label3.setVerticalTextPosition( SwingConstants.BOTTOM );
42          label3.setToolTipText( "This is label3" );
43          container.add( label3 );
44
45          setSize( 275, 170 );
46          setVisible( true );
47       }
48
49       // execute application
50       public static void main( String args[] )
51       {
52          LabelTest application = new LabelTest();
53
54          application.setDefaultCloseOperation(
55             JFrame.EXIT_ON_CLOSE );
56       }
57
58    }  // end class LabelTest
```





يقوم البرنامج بتعريف ثلاثة عناصر (كائنات) من نوع JLabel في السطر ١٢. ثم قمنا بإعطاء العناصر قيم في الباني LabelTest Constructor (السطر ١٥ -٤٧). السطر ٢٤ ينشئ عنصر JLabel يحتوي على النص "Label with text". ويظهر هذا النص عند إظهار نافذة التنفيذ على الشاشه.

السطر ٢٥ يستخدم الطريقة setToolTipText لتحديد الملحظه التي ستظهر عند تحريك المؤشر فوق هذا العنصر.

السطر ٢٦ يضيف العنصر label1 إلى شريط المحتويات content pane. العديد من العناصر في الحزمة swing يمكنها إظهار صور وذلك بتحديد عنصر من نوع Icon كباراميتر في منشيء العنصر أو باستخدام طريقة setIcon.

إحدى الفصائل التي تطبق الواجهة أيقون Icon interface هي الفصيله ImageIcon والتي تدعم العديد من تنسيقات الصور image formats مثل: GIF, PNG و GPEG. السطر ٣٠ يعرف الكائن من نوع ImageIcon. أما الملف bug1.gif فيحتوي على الصورة التي ستحمل وتخزن في الكائن ImageIcon، كما أننا نفترض أن ملف الصورة موجود في نفس الدليل folder الذي يحتوي على البرنامج. ان الكائن ImageIcon تم إسناده إلى متغير مرجعي من نوع أيقون Icon اسمه bug. تذكر إن الفصيلة ImageIcon تطبق الواجهة Icon وبالتالي فإن ImageIcon هو Icon.

الفصيلة JLabel تدعم عملية عرض الأيقونات، السطر ٣١ -٣٢ يستخدم منشئاً آخر للفصيل JLabel لإظهار label يعرض النص "Label with text and Icon" والأيقونة التي يشير إليها المتغير bug محدد اتجاه ليكون على الطرف اليسار من Label.حيث تم تحديد الاتجاه باستخدام ثابت عددي من خلال المعرف SwingConstants.LEFT. لاحظ أن الوضع التلقائي في حالة وجود نص مع أيقونة على نفس العنصر الرسومي label أن تكون الأيقونة على يمين النص. كما يمكنك تحديد التنسيقات الأفقية والعمودية للعنصر الرسومي label من خلال الطرق setHorizontalAlignment و setVerticalAlignment. السطر ٣٣ يحدد الملحظة tool tip للعنصر label2. والسطر ٣٤ يضيف هذا العنصر إلى content pane. السطر ٣٧ ينشئ كائناً من نوع label باستخدام باني من غير أي بارميتر وفي هذه الحالة فإن هذا الكائن لا يحتوي على نص أو أيقونة بداخله. السطر ٣٨ يستخدم الطريقة setText لتحديد النص للكائن label3 كما ويوجد هناك طريقة أخرى لاسترجاع النص من label وهي getText. السطر ٣٩ يستخدم الطريقة setIcon لتحديد الأيقونة للكائن label3 كما وتوجد هناك طريقه لاسترجاع الأيقونة الموجودة في الكائن وهي getIcon. الأسطر ٤٠ -٤١ تستخدم الطرق setHorizontalTextPosition و setVerticalTextPosition لتحديد موقع النص في

العنصر الرسومي label. ففي هذا البرنامج سيكون النص أفقيا في الوسط وعموديا في الأسفل بمعنى أن الأيقونة ستكون في الأعلى.

السطر ٤٢ يحدد الملحظه للكائن الرسومي label3 والسطر ٤٣ يضيف label3 إلى شريط المحتويات content pane.

العنصر الرسومي JTextField **و العنصر الرسومي** JPasswordField.

العنصران JTextField و JPasswordField هي مناطق أحادية السطر تستخدم لإدخال نص من قبل المستخدم عن طريق لوحة المفاتيح. في حين أن العنصر JPasswordField يظهر أن أحرف قد تم إدخالها دون إظهار الأحرف نفسها. عندما يدخل المستخدم البيانات في أحد هذين العنصرين ثم ضغط enter يتسبب ذلك في إطلاق حدث وفي حالة تسجيل أحد العنصرين أو كلاهما في مستمع للحدث event listener فانه يمكن معالجة الحدث واستخدام النص من العنصر الرسومي.

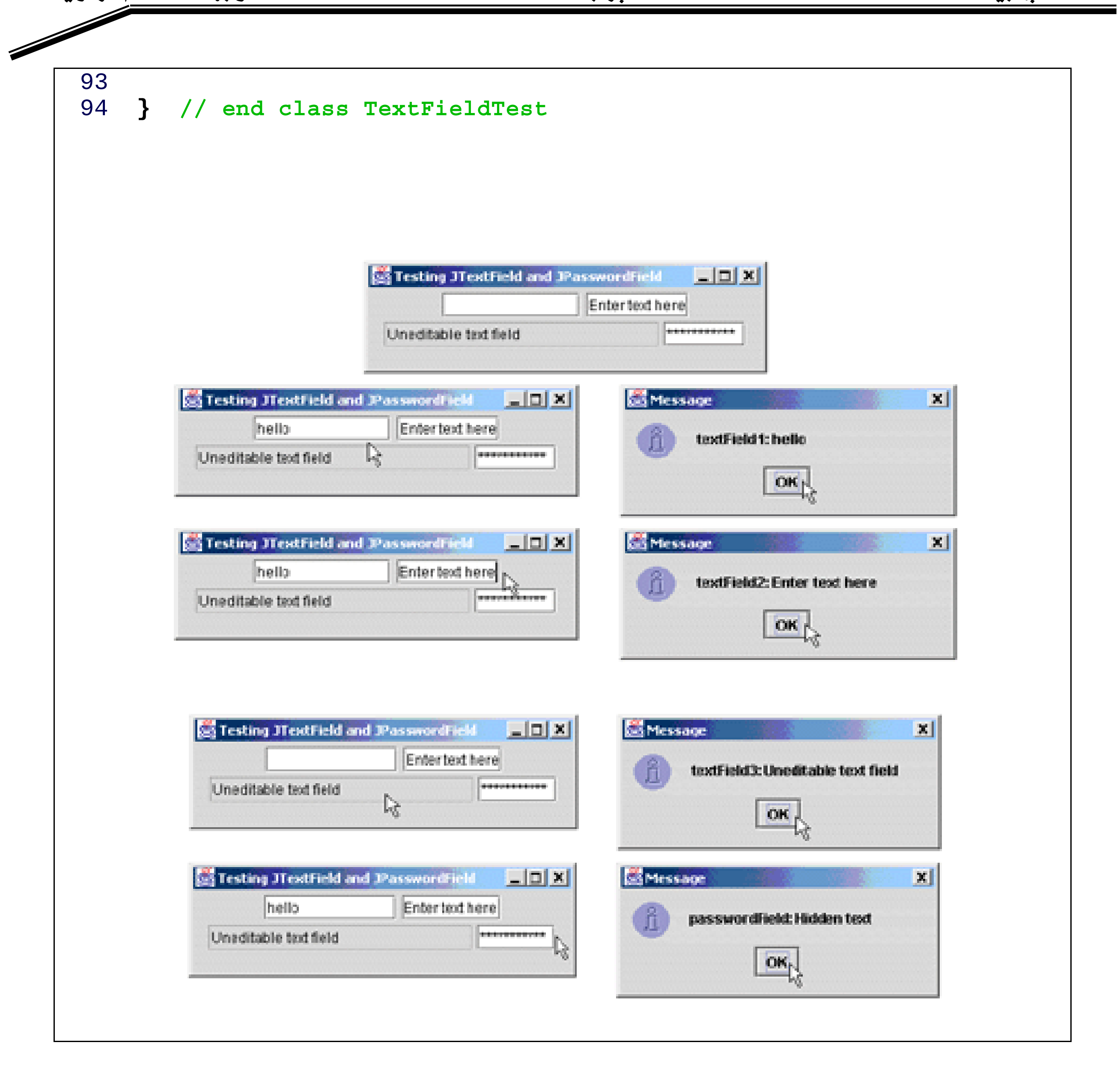
البرنامج التالي يوضح استخدام العنصرين الرسوميين JTextField و JPasswordField والطرق الخاصه بهما:

```
1
2    // Demonstrating the JTextField class.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class TextFieldTest extends JFrame {
12      private JTextField textField1, textField2, textField3;
13      private JPasswordField passwordField;
14
15      // set up GUI
16      public TextFieldTest()
17      {
18        super( "Testing JTextField and JPasswordField" );
19
20        Container container = getContentPane();
21        container.setLayout( new FlowLayout() );
22
23        // construct textfield with default sizing
24        textField1 = new JTextField( 10 );
25        container.add( textField1 );
26
27        // construct textfield with default text
28        textField2 = new JTextField( "Enter text here" );
29        container.add( textField2 );
30
31        // construct textfield with default text and
32        // 20 visible elements and no event handler
33     textField3 = new JTextField( "Uneditable text field", 20 );
34        textField3.setEditable( false );
35        container.add( textField3 );
36
37        // construct textfield with default text
38        passwordField = new JPasswordField( "Hidden text" );
```

```
39         container.add( passwordField );
40
41         // register event handlers
42         TextFieldHandler handler = new TextFieldHandler();

43         textField1.addActionListener( handler );
44         textField2.addActionListener( handler );
45         textField3.addActionListener( handler );
46         passwordField.addActionListener( handler );
47
48           setSize( 325, 100 );
49        setVisible( true );
50     }
51
52     // execute application
53     public static void main( String args[] )
54     {
55        TextFieldTest application = new TextFieldTest();
56
57        application.setDefaultCloseOperation(
58           JFrame.EXIT_ON_CLOSE );
59     }
60
61     // private inner class for event handling
62     private class TextFieldHandler implements ActionListener {
63
64        // process text field events
65        public void actionPerformed( ActionEvent event )
66        {
67           String string = "";
68
69           // user pressed Enter in JTextField textField1
70           if ( event.getSource() == textField1 )
71              string = "textField1: " + event.getActionCommand();
72
73           // user pressed Enter in JTextField textField2
74           else if ( event.getSource() == textField2 )
75              string = "textField2: " + event.getActionCommand();
76
77           // user pressed Enter in JTextField textField3
78           else if ( event.getSource() == textField3 )
79              string = "textField3: " + event.getActionCommand();
80
81           // user pressed Enter in JTextField passwordField
82           else if ( event.getSource() == passwordField ) {
83              JPasswordField pwd =
84                 ( JPasswordField ) event.getSource();
85              string = "passwordField: " +
86                  new String( passwordField.getPassword() );
87           }
88
89           JOptionPane.showMessageDialog( null, string );
90        }
91
92     }  // end private inner class TextFieldHandler
```

```
93
94  }  // end class TextFieldTest
```



السطر١٢ -١٣ يعرف ثلاثة عناصر من نوع JTextField و عنصر من نوع JPasswordField، وكلا من هذه العناصر قد تم انشاؤه من خلال البيانات constructors في الأسطر ١٦ -٥٠. السطر ٢٤ يعرف الحقل النصي textField1 بطول ١٠ خانات. السطر ٢٥ يضيف الحقل النصي textField1 إلى شريط المحتويات content pane.

السطر ٢٨ يعرف حقل نصي آخر هو textField2 مع نص أولي "Enter Text Here" ليظهر في الحقل النصي. لاحظ أننا لم نحدد طول الحقل النصي حيث إن طول هذا الحقل سيكون مساوياً لطول النص بداخله. السطر ٢٩ يضيف الحقل النصي إلى شريط المحتويات content pane.

السطر ٣٣ يعرف حقل نصي آخر هو textField3 وينادي باني الفصيلة JTextField مع باراميترين هما النص الأولي "Uneditable Text field" ,وعدد خانات(٢٠) يمثل طول الحقل النصي. السطر ٣٤ يستخدم الطريقة setEditable والباراميتر false ، لتحديد أن المستخدم لا يمكنه تعديل محتويات الحقل. السطر ٣٥ يضيف الحقل النصي إلى شريط المحتويات content pane.

السطر ٣٨ يعرف عنصر passwordField من نوع PasswordField مع النص "Hidden Text" ليظهر في داخل الحقل، وطول هذا العنصر يحدد من خلال طول النص المحدد داخل الحقل. لاحظ أن النص المدخل سيظهر على شكل مجموعة من النجوم asterisks كنوع من الأمان. السطر ٣٩ يضيف حقل كلمة المرور إلى شريط المحتويات content pane.

ولمعالجة الأحداث في هذا البرنامج قمنا بإنشاء صنف خاص TextFieldHandler داخل الفصيلة الرئيسة (السطر ٦٢ -٩٢) والذي يرث الواجهة interface المسماة ActionListener وبالتالي فإن كل عنصر من نوع TextFieldHandler هو عنصر من نوع ActionListener . السطر ٤٢ يعرف عنصر مرجعي من فصيلة TextFieldHandler اسمه Handler، والذي سيستخدم كمستمع للأحداث event-listener على عناصر الحقول النصية وعنصر حقل كلمة المرور.

الأسطر ٤٣ -٤٦ تحتوي على جمل تسجيل عناصر الحقول النصية وكلمة المرور في المستمع handler بعد تنفيذ هذه الجمل، فإن أي حدث (الضغط على زر enter) على العناصر المسجلة في المستمع سيؤدي إلى إطلاق الطريقة actionPerformed.

تقوم الطريقة actionPerformed بتحديد العنصر الذي تسبب في إطلاقها وذلك باستخدام الطريقة getSource الموجودة داخل الفصيلة ActionEvent، وبعد تحديد العنصر نقوم ببناء جملة نصية وتخزينها في متغير اسمه string من نوع String استخدمنا فيها الطريقة getActionCommand الموجودة في الصنف ActionEvent والتي تستخدم لاسترجاع النص من عنصر حرفي TextField والطريقة getPassword الخاصة بالصنف JPasswordField لاسترجاع النص من عنصر كلمة المرور.

العنصر الرسومي (زر) JButton

الزر هو عنصر رسومي يضغطه المستخدم لإطلاق عمل معين، يمكنك عند كتابة برنامج جافا استخدام ازرار من عدة أنواع مثل:

command buttons, check boxes, radio buttons، وسنتطرق للأنواع الثلاثة في الصفحات التالية ونعرض مثالاً على كل منها.

المثال التالي يوضح استخدام زر الأمر Command Buttons. هذا النوع من الأزرار يسبب أطلاق حدث من نوع ActionEvent عند الضغط عليه بالفأرة من قبل المستخدم. ويتم إنشاء عنصر زر الأمر من خلال الصنف JButton.

```
1
2    // Creating JButtons.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class ButtonTest extends JFrame {
12      private JButton plainButton, fancyButton;
13
14      // set up GUI
15      public ButtonTest()
16      {
17         super( "Testing Buttons" );
18
19         // get content pane and set its layout
20         Container container = getContentPane();
21         container.setLayout( new FlowLayout() );
22
23         // create buttons
24         plainButton = new JButton( "Plain Button" );
25         container.add( plainButton );
26
27         Icon bug1 = new ImageIcon( "bug1.gif" );
28         Icon bug2 = new ImageIcon( "bug2.gif" );
29         fancyButton = new JButton( "Fancy Button", bug1 );
30         fancyButton.setRolloverIcon( bug2 );
31         container.add( fancyButton );
32
33         // create an instance of inner class ButtonHandler
34         // to use for button event handling
35         ButtonHandler handler = new ButtonHandler();
36         fancyButton.addActionListener( handler );
37         plainButton.addActionListener( handler );
```

```
38
39          setSize( 275, 100 );

40          setVisible( true );
41       }
42
43       // execute application
44       public static void main( String args[] )
45       {
46          ButtonTest application = new ButtonTest();
47
48          application.setDefaultCloseOperation(
49             JFrame.EXIT_ON_CLOSE );
50       }
51
52       // inner class for button event handling
53       private class ButtonHandler implements ActionListener {
54
55          // handle button event
56          public void actionPerformed( ActionEvent event )
57          {
58             JOptionPane.showMessageDialog( null,
59                "You pressed: " + event.getActionCommand() );
60          }
61
62       }  // end private inner class ButtonHandler
63
64    }  // end class ButtonTest
```

المثال أعلاه يقوم بانشاء عنصرين من نوع JButtons، العناصر من هذا النوع يمكنها إظهار الأيقونات إلى جانب إظهارها للنص كما هو الحال في العناصر من نوع JLabel. معالجة الأحداث للأزرار تتم من خلال كائن من نوع الصنف الداخلي inner class المسمى ButtonHandler (الأسطر ٥٣ -٦٢).

السطر ١٢ يعرف العنصرين من نوع أزرار الأوامر JButton هما: plainButton و fancyButton واللذان تم اعطاؤهما القيم المبدئية في باني الصنف.

السطر ٢٤ ينشئ plainButton ويعطيه النص "plain button" كعنوان لهذا الزر. السطر ٢٥ يضيف العنصر على شريط المحتويات content pane.

يستطيع صنف الأزرار JButton إظهار الأيقونات على الأزرار لتحسين شكل الواجهة الرسومية، كما أن هذا الصنف يوفر خاصية أيقونة المرور على الزر rollover icon وهي الأيقونة التي ستظهر وتختفي على الزر عند مرور الفأرة فوق الزر وخارجه. السطر ٢٧ -٢٨ ينشئ عنصرين من نو ع ImageIcon يمثلان أيقونة الزر الرئيسة وأيقونة المرور على الزر rolover icon للزر المنشأ في السطر ٢٩. كلا السطرين يفترضان أن ملفات الصور مخزنة في نفس الدليل المخزن فيه برنامج جافا.

السطر ٢٩ ينشئ الزر fancyButton مع نص مبدئي هو " Fancy Button " والأيقونة bug1. الوضع التلقائي أن يكون النص على يمين الأيقونة. السطر ٣٠ يستخدم الطريقة setRolloverIcon لتحديد الأيقونة التي ستظهر على الزر عند وضع الفأرة على الزر. السطر ٣١ يضيف الزر إلى شريط المحتويات content pane.

إن الأزرار JButton (مثل الحقول النصيه JTextField) يولد ActionEvent، السطر ٣٥ -٣٧ يسجل كائن مستمع listener لكل زر في البرنامج. السطر ٥٣ -٦٢ يعرف صنف داخلي ButtonHandler والذي يحتوي على الطريقة actionPerformed، تقوم هذه الطريقة بإظهار صندوق رسالة يحتوي على النص الموجود داخل الزر الذي تم معالجته من قبل المستخدم.

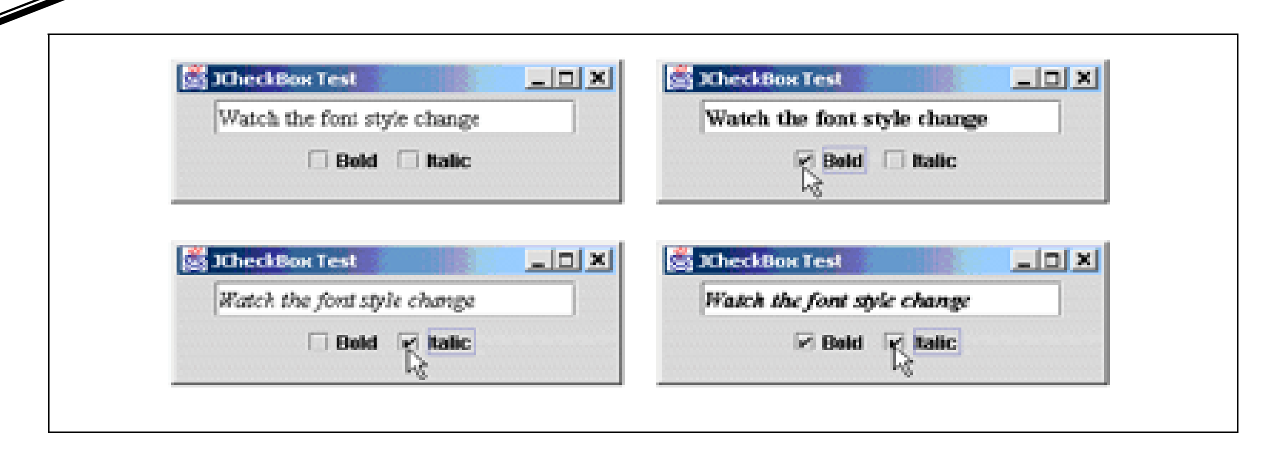
العنصر الرسومي JCheckBox.

المثال التالي يوضح كيفية تعريف واستخدام العنصر الرسومي JCheckBox، يقوم هذا البرنامج بتعريف عنصرين من نوع JCheckBox لتغير شكل الخط (أسود، مائل) المكتوب في حقل نصي JTextField فإذا تم اختيار صندوق شكل الخط أسود سيتحول الخط إلى الأسود، وإذا تم اختيار صندوق مائل سيتحول النص إلى مائل، وإذا تم اختيار الصندوقين سيتحول النص إلى أسود ومائل. عند بداية تنفيذ البرنامج لن يكون أي من الصندوقين في وضع الاختيار.

```
1
2    // Creating Checkbox buttons.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class CheckBoxTest extends JFrame {
12      private JTextField field;
13      private JCheckBox bold, italic;
14
15      // set up GUI
16      public CheckBoxTest()
17      {
18         super( "JCheckBox Test" );
19
20         // get content pane and set its layout
21         Container container = getContentPane();
22         container.setLayout( new FlowLayout() );
23
24         // set up JTextField and set its font
25         field =
26            new JTextField( "Watch the font style change", 20 );
27         field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
28         container.add( field );
29
30         // create checkbox objects
31         bold = new JCheckBox( "Bold" );
32         container.add( bold );
33
34         italic = new JCheckBox( "Italic" );
35         container.add( italic );
36
37         // register listeners for JCheckBoxes
38         CheckBoxHandler handler = new CheckBoxHandler();
39         bold.addItemListener( handler );
40         italic.addItemListener( handler );
41
42         setSize( 275, 100 );
43         setVisible( true );
```

```
44        }
45
46        // execute application
47        public static void main( String args[] )
48        {
49           CheckBoxTest application = new CheckBoxTest();
50
51           application.setDefaultCloseOperation(
52              JFrame.EXIT_ON_CLOSE );
53        }
54
55        // private inner class for ItemListener event handling
56        private class CheckBoxHandler implements ItemListener {
57           private int valBold = Font.PLAIN;
58           private int valItalic = Font.PLAIN;
59
60           // respond to checkbox events
61           public void itemStateChanged( ItemEvent event )
62           {
63              // process bold checkbox events
64              if ( event.getSource() == bold )
65
66                 if ( event.getStateChange() == ItemEvent.SELECTED )
67                    valBold = Font.BOLD;
68                 else
69                    valBold = Font.PLAIN;
70
71              // process italic checkbox events
72              if ( event.getSource() == italic )
73
74                 if ( event.getStateChange() == ItemEvent.SELECTED )
75                    valItalic = Font.ITALIC;
76                 else
77                    valItalic = Font.PLAIN;
78
79              // set text field font
80              field.setFont(
81                 new Font( "Serif", valBold + valItalic, 14 ) );
82           }
83
84        }  // end private inner class CheckBoxHandler
85
86     }  // end class CheckBoxTest
```





بعد إنشاء الحقل النصي وتحديد النص المبدأي بداخله، السطر ٢٧ قام بتحديد التنسيق للخط ليكون نوعه serif، وشكله عادي PLAIN و حجمه ١٤ نقطة. ثم يقوم باني الفصيل بإنشاء عنصرين من نوع JCheckBox في الأسطر ٣١ -٣٤. إن النص الحرفي المرسل كباراميتر لباني الفصيل يمثل النص الذي سيظهر على يمين الحقل JCheckBox.

عندما يقوم المستخدم بالضغط على أحد العنصرين، يؤدي ذلك إلى أطلاق حدث من نوع ItemEvent والذي يمكن معالجته من خلال المستمع ItemListener، ويتوجب على هذا المستمع تعريف الطريقة itemStateChanged لبرمجة المعالجة الخاصة بهذا العنصر.

يتم مناداة الطريقة itemStateChanged في حالة قام المستخدم بالضغط على أحد العنصرين bold أو italic. حيث تستخدم الطريقة event.getSource() لتحديد أي العنصرين تم الضغط عليه. ففي حال أنه العنصر bold فإن جملة if/else في الأسطر ٦٦ -٦٩ تستخدم الطريقة getStateChanged المعرفة في الفصيل ItemEvent لتحديد حالة العنصر: هل هو في حالة اختيار أم عدم اختيار؟ (ItemEvent.SELECTED أو ItemEvent.DESELECTED) اذا كانت الحالة اختيار فإن القيمة العددية للثابت Font.BOLD يتم إسنادها للمتغير العددي valBold وإلا فإن القيمة العددية للثابت Font.PLAIN تسند له. نفس جملة الشرط تعاد للعنصر italic بحيث إنه إذا كانت حالة العنصر اختيار فإن القيمة العددية للثابت Font.ITALIC تسند للمتغير العددي valItalic والا فإن القيمة العددية للثابت Font.PLAIN تسند له. مجموع الحالتين valBold و valItalic استخدمت في الأسطر ٨٠ -٨١ كشكل للخط في الحقل النصي JTextField.

العنصر الرسومي JRadioButton.

تتشابه العناصر الرسومية من نوع JRadioButton مع العناصر من نوع JCheckBox في كون كل منها له حالتان مختار، وغير مختار (selected and deselected) إلا أن radio buttons تظهر غالبا على شكل مجموعة بحيث إن أحد عناصرها يتم اختياره فقط والباقي غير مختار. فعند الضغط على خيار آخر في المجموعة فإن الخيار الأول يتم إلغاؤه deselected. ولجمع عدد من JRadioButtons في مجموعة واحدة سوف نستخدم كائناً من نوع ButtonGroup والذي لا يعتبر عنصر رسومي(على الرغم من وجوده في الحزمة javax.swing) فهو لا يظهر على الشاشة، ووظيفته تنحصر في تحديد العناصر من نوع JRadioButtons التي تمثل مجموعة واحدة.

المثال التالي شبيه بالمثال الخاص بالعنصر الرسومي JCheckBox أعلاه، حيث يستطيع المستخدم تغيير تنسيق الخط في الحقل النصي. يستخدم هذا البرنامج radio buttons لتطبيق تنسيق واحد فقط على النص.

```
1
2    // Creating radio buttons using ButtonGroup and JRadioButton.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class RadioButtonTest extends JFrame {
12    private JTextField field;
13    private Font plainFont, boldFont, italicFont, boldItalicFont;
14    private JRadioButton plainButton, boldButton, italicButton,
15            boldItalicButton;
16    private ButtonGroup radioGroup;
17
18      // create GUI and fonts
19      public RadioButtonTest()
20      {
21         super( "RadioButton Test" );
22
23         // get content pane and set its layout
24         Container container = getContentPane();
25         container.setLayout( new FlowLayout() );
26
27         // set up JTextField
28         field =
29            new JTextField( "Watch the font style change", 25 );
30         container.add( field );
31
```

```
32          // create radio buttons
33          plainButton = new JRadioButton( "Plain", true );
34          container.add( plainButton );
35
36          boldButton = new JRadioButton( "Bold", false);
37          container.add( boldButton );
38
39          italicButton = new JRadioButton( "Italic", false );
40          container.add( italicButton );
41
42          boldItalicButton = new JRadioButton(
43             "Bold/Italic", false );
44          container.add( boldItalicButton );
45
46          // register events for JRadioButtons
47          RadioButtonHandler handler = new RadioButtonHandler();
48          plainButton.addItemListener( handler );
49          boldButton.addItemListener( handler );
50          italicButton.addItemListener( handler );
51          boldItalicButton.addItemListener( handler );
52
53          // create logical relationship between JRadioButtons
54          radioGroup = new ButtonGroup();
55          radioGroup.add( plainButton );
56          radioGroup.add( boldButton );
57          radioGroup.add( italicButton );
58          radioGroup.add( boldItalicButton );
59
60          // create font objects
61          plainFont = new Font( "Serif", Font.PLAIN, 14 );
62          boldFont = new Font( "Serif", Font.BOLD, 14 );
63          italicFont = new Font( "Serif", Font.ITALIC, 14 );
64          boldItalicFont =
65             new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
66          field.setFont( plainFont );
67
68          setSize( 300, 100 );
69          setVisible( true );
70       }
71
72       // execute application
73       public static void main( String args[] )
74       {
75          RadioButtonTest application = new RadioButtonTest();
76
77          application.setDefaultCloseOperation(
78             JFrame.EXIT_ON_CLOSE );
79       }
80
81       // private inner class to handle radio button events
82       private class RadioButtonHandler implements ItemListener {
83
84          // handle radio button events
85          public void itemStateChanged( ItemEvent event )
86          {
```

```
87              // user clicked plainButton
88              if ( event.getSource() == plainButton )
89                 field.setFont( plainFont );
90
91              // user clicked boldButton
92              else if ( event.getSource() == boldButton )
93                 field.setFont( boldFont );
94
95              // user clicked italicButton
96              else if ( event.getSource() == italicButton )
97                 field.setFont( italicFont );
98
99              // user clicked boldItalicButton
100             else if ( event.getSource() == boldItalicButton )
101                field.setFont( boldItalicFont );
102          }
103
104       }  // end private inner class RadioButtonHandler
105
106    }  // end class RadioButtonTest
```

السطر ٣٣ -٤٤ يعرف كل عنصر من عناصر JRadioButton ويضيفه إلى شريط المحتويات content pane كل كائن من هذه الكائنات تم إنشاؤه وإعطاؤه قيماً من خلال بانيات الفصيل كما في السطر ٣٣، هذا الباني يزود كل عنصر من JRadioButton بعنوان (label) يظهر إلى يمين العنصر، و حالة العنصر. حيث إن القيمه true تعني أن هذا العنصر يجب أن يظهر في الاختيار select. عناصر JRadioButtons مثل عناصر JCheckBox تطلق حدثاً من نوع ItemEvent عندما يتم الضغط عليها. الأسطر ٤٧ -٥١ ينشئ كائن من الفصيل الداخلي RadioButtonHandler (والمعرف في الأسطر ٨٢ -١٠٤) وتسجيله لمعالجة الأحداث ItemEvent التي ستنطلق عند ضغط المستخدم على أي من عناصر JRadioButtons.

السطر ٥٤ يعرف كائناً من نوع ButtonGroup اسمه radioGroup، سيستخدم هذا الكائن لربط العناصر من نوع JRadioButton في مجموعة واحدة بحيث يتم اختيار واحد فقط من هذه العناصر في الوقت الواحد. الأسطر ٥٥ -٥٨ تستخدم الطريقة add المعرفة داخل الفصيل ButtonGroup لربط كل عنصر من نوع JRadioGroup بالمجموعة المسماة radioGroup.

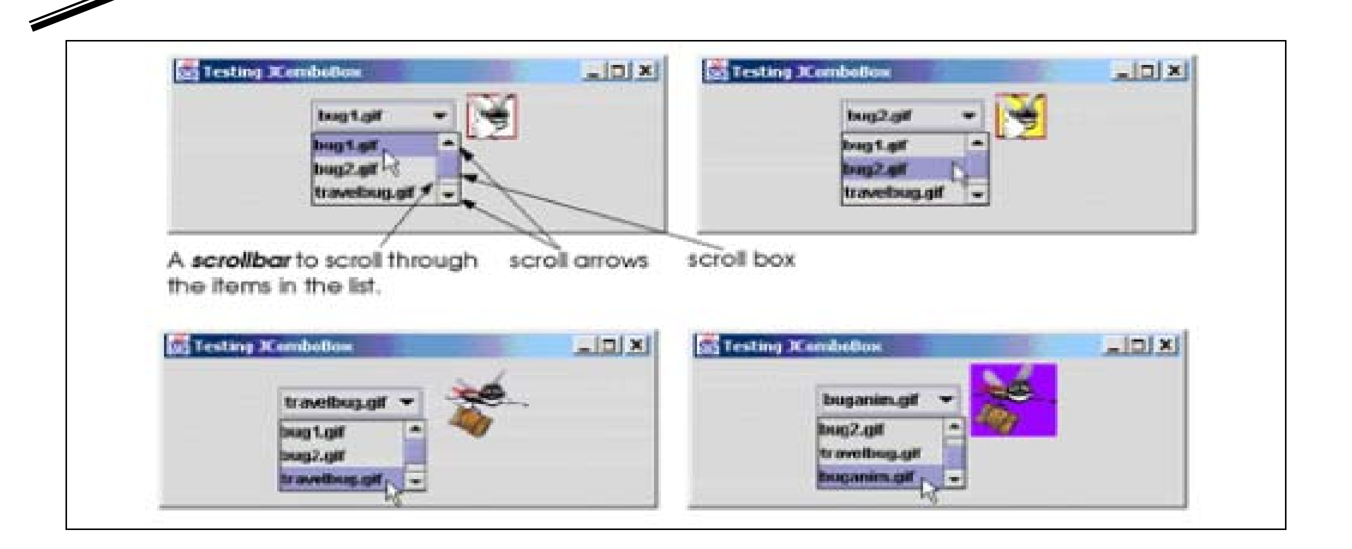
الفصيل RadioButtonHandler (الأسطر ٨٢ -١٠٤) يطبق الواجهة ItemListener وبالتالي فإنه يمكنه معالجة الأحداث من نوع ItemEvent الناتجة عن عناصر JRadioButton. عند الضغط على أي عنصر JRadioButton فإن المجموعة radioGroup تلغي اختيار العنصر السابق وتختار العنصر الحالي وتنفذ الطريقة itemStateChanged (الأسطر ٨٥ -١٠٢) حيث تقوم بتحديد العنصر الذي تم الضغط عليه باستخدام الطريقة getSource، ثم تغير تنسيق الحقل النصي إلى التنسيق الجديد.

العنصر الرسومي JComboBox

عنصر القوائم JComboBox يوفر إمكانية عمل قائمة من الخيارات يستطيع المستخدم الاختيار منها. عنصر القوائم JComboBox مثله مثل العنصرين JCheckBox و JRadioButton يتسبب في إطلاق الحدث ItemEvent عند الضغط عليه. المثال التالي يوضح كيفية تعريف واستخدام عنصر القوائم.

```
1
2    // Using a JComboBox to select an image to display.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class ComboBoxTest extends JFrame {
12      private JComboBox imagesComboBox;
13      private JLabel label;
14
15      private String names[] =
16         { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
17      private Icon icons[] = { new ImageIcon( names[ 0 ] ),
18         new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
19         new ImageIcon( names[ 3 ] ) };
20
21      // set up GUI
22      public ComboBoxTest()
23      {
24         super( "Testing JComboBox" );
```

```
25
26          // get content pane and set its layout
27          Container container = getContentPane();
28          container.setLayout( new FlowLayout() );
29
30          // set up JComboBox and register its event handler
31          imagesComboBox = new JComboBox( names );
32          imagesComboBox.setMaximumRowCount( 3 );
33
34          imagesComboBox.addItemListener(
35
36             // anonymous inner class to handle JComboBox events
37             new ItemListener() {
38
39                // handle JComboBox event
40                public void itemStateChanged( ItemEvent event )
41                {
42                   // determine whether check box selected
43                   if ( event.getStateChange() == ItemEvent.SELECTED )
44                      label.setIcon( icons[
45                         imagesComboBox.getSelectedIndex() ] );
46                }
47
48             }  // end anonymous inner class
49
50          ); // end call to addItemListener
51
52          container.add( imagesComboBox );
53
54          // set up JLabel to display ImageIcons
55          label = new JLabel( icons[ 0 ] );
56          container.add( label );
57
58          setSize( 350, 100 );
59          setVisible( true );
60       }
61
62       // execute application
63       public static void main( String args[] )
64       {
65          ComboBoxTest application = new ComboBoxTest();
66
67          application.setDefaultCloseOperation(
68             JFrame.EXIT_ON_CLOSE );
69       }
70
71    }  // end class ComboBoxTest
```



يستخدم هذا المثال عنصر القوائم JComboBox لتوفير قائمة من أربع خيارات تمثل أسماء ملفات من نوع صور، وعند الضغط على خيار ما في القائمة، ستظهر الصورة الموجودة في الملف على شكل أيقونة Icon داخل عنصر رسومي من نوع JLabel .

السطر ١٧ -١٩ يعرف مصفوفة اسمها icons ويعطيها القيم الأولية، تحتوي المصفوفة على اربع كائنات من نوع ImageIcon، كما يعرف مصفوفه أخرى من نوع String اسمها names تحتوي على أسماء ملفات الصور المخزنة في نفس الدليل الذي يحتوي على البرنامج.

السطر ٣١ ينشئ كائناً من نوع قائمة JComboBox ويستخدم عناصر المصفوفة names كعناصر للقائمه. مؤشر عددي يتابع ترتيب العناصر في القائمة. العنصر الأول يضاف في الموقع (٠) في القائمة، العنصر الثاني يضاف في الموقع (١) من القائمة وهكذا. العنصر الأول في القائمة يظهر في وضعية الاختيار عند إظهار القائمة على الشاشة. باقي العناصر يتم اختيارها بالضغط عليها من القائمة.

السطر ٣٢ يستخدم الطريقة setMaximumRowCount المعرفة في الفصيلة JComboBox لتحديد الحد الأقصى من العناصر التي ستظهر عند الضغط على القائمه. وفي حالة وجود عدد من العناصر أكبر من الحد الأقصى الممكن إظهاره فإنه يظهر في القائمه شريط تصفح عمودي لتمكين المستخدم من إظهار العناصر المتبقية. السطر ٣٤ -٥٠ يسجل كائناً من نوع الفصيل الداخلي (بدون اسم) والذي يطبق الواجهة ItemListener، حيث تم تسجيله كمستمع على القائمه imagesComboBox. فعندما يختار المستخدم أحد العناصر من القائمه فإن الطريقة itemStateChanged تنفذ (الأسطر ٤٠ -٤٨)

وتقوم بوضع الأيقونة لحقل label. يتم اختيار الأيقونة من مصفوفة icons بعد تحديد موقع العنصر بالمصفوفة بواسطة الطريقة getSelectedIndex في السطر ٤٥.

مديرو عرض العناصر الرسومية Layout Managers

تم تزويد العناصر الرسومية GUI Components الموجودة ضمن حأويات Containers بمديري عرض العناصر الرسومية وذلك لأغراض عرض العناصر في واجهة التطبيق داخل الحأوية Container بشكل منسق. الشكل التالي يوضح ثلاثة أنواع من مديري عرض العناصر الرسومية:

مدير العرض	الوصف
FlowLayout	هذا العرض هو العرض التلقائي لكل من java.awt.Applet, java.awt.Panel, javax.swing.Jpanel. ويقوم بعرض العناصر الرسومية بشكل متسلسل حسب ترتيب إدراجها في حأوية العناصر.
BorderLayout	يستخدم كعرض تلقائي لشريط المحتويات Content pane الخاص بـ JFrame و JApplet، يقوم بترتيب العناصر في خمس مناطق: الشمالية، الجنوبية، الشرقية، الغربية والوسطى.
GridLayout	يقوم هذا العارض بترتيب العناصر في سطور وأعمده.

واليك الأمثلة التالية لتوضيح كل من الأنواع الثلاثة أعلاه.

مدير العرض FlowLayout

```
1
2     // Demonstrating FlowLayout alignments.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
7
8     // Java extension packages
9     import javax.swing.*;
10
11    public class FlowLayoutDemo extends JFrame {
12       private JButton leftButton, centerButton, rightButton;
13       private Container container;
14       private FlowLayout layout;
```

```
15
16       // set up GUI and register button listeners
17       public FlowLayoutDemo()
18       {
19          super( "FlowLayout Demo" );
20
21          layout = new FlowLayout();
22
23          // get content pane and set its layout

24          container = getContentPane();
25          container.setLayout( layout );
26
27          // set up leftButton and register listener
28          leftButton = new JButton( "Left" );
29
30          leftButton.addActionListener(
31
32             // anonymous inner class
33             new ActionListener() {
34
35                // process leftButton event
36                public void actionPerformed( ActionEvent event )
37                {
38                   layout.setAlignment( FlowLayout.LEFT );
39
40                   // re-align attached components
41                   layout.layoutContainer( container );
42                }
43
44             }  // end anonymous inner class
45
46          ); // end call to addActionListener
47
48          container.add( leftButton );
49
50          // set up centerButton and register listener
51          centerButton = new JButton( "Center" );
52
53          centerButton.addActionListener(
54
55             // anonymous inner class
56             new ActionListener() {
57
58                // process centerButton event
59                public void actionPerformed( ActionEvent event )
60                {
61                   layout.setAlignment( FlowLayout.CENTER );
62
63                   // re-align attached components
64                   layout.layoutContainer( container );
65                }
66             }
67          );
68
69          container.add( centerButton );
```

```
70
71          // set up rightButton and register listener
72          rightButton = new JButton( "Right" );
73
74          rightButton.addActionListener(
75
76             // anonymous inner class
77             new ActionListener() {
78
79                // process rightButton event
80                public void actionPerformed( ActionEvent event )
81                {

82                   layout.setAlignment( FlowLayout.RIGHT );
83
84                   // re-align attached components
85                   layout.layoutContainer( container );
86                }
87             }
88          );
89
90          container.add( rightButton );
91
92          setSize( 300, 75 );
93          setVisible( true );
94       }
95
96       // execute application
97       public static void main( String args[] )
98       {
99          FlowLayoutDemo application = new FlowLayoutDemo();
100
101         application.setDefaultCloseOperation(
102            JFrame.EXIT_ON_CLOSE );
103      }
104
105   }  // end class FlowLayoutDemo
```

FlowLayout Demo — Left | Center | Right

البرنامج أعلاه يرسم ثلاثة ازرار من نوع JButtons ويضيفهم إلى التطبيق باستخدام FlowLayout يتم وضع العناصر في الوسط تلقائيا، وعند الضغط على زرLeft يتحول وضع العناصر لتبدأ من اليسار وعند الضغط على زر Right يتحول وضع العناصر لتبدأ من اليمين، وكذلك عند الضغط على زر Center يتم توسيط العناصر. لاحظ إنه عند تصغير عرض النافذة فإن الزر الثالث لا يعود له مكان على نفس السطر لذلك سينتقل إلى سطر جديد.

كما تلاحظ في السطر ٢٥ فإنه يتم تحديد Layout للحاوية Container من خلال الطريقة setLayout، كما تلاحظ إنه يمكنك تغيير وضع العناصر الرسومية ابتداء من اليمين أو اليسار أو الوسط، من خلال الطريقة Layout.setAlignment.

مدير العرض BorderLayout

يقوم هذا العارض بتقسيم الحأويه Container إلى خمس مناطق هي: شمالية، جنوبية، شرقية، ووسطى. يمكنك إضافة عنصر واحد لكل من هذه المناطق الخمس. هذا العنصر يمكن أن يكون حأوية container يحتوي على العديد من العناصر بداخله. المنطقة الشمالية والجنوبية تمتد أفقياً حتى نهاية أطراف الحاوية. اما المنطقة الشرقية والغربية فتمتد عموديا بين المنطقتين الشمالية والجنوبية. أما المنطقة المتبقية فهي للمنطقة الوسطى. في حال عدم وجود المنطقة الشمالية والجنوبية فإن كلا من المناطق الشرقية، الوسطى، والغربية تتمدد لتغطية المنطقه الفارغة. وفي حالة عدم وجود المناطق الشرقية والغربية فإن المنطقة الوسطى تتمدد لتغطية المنطقة الفارغة.المثال التالي يوضح استخدام العارض BorderLayout :

```
1
2   // Demonstrating BorderLayout.
3
4   // Java core packages
5   import java.awt.*;
6   import java.awt.event.*;
7
8   // Java extension packages
9   import javax.swing.*;
10
11  public class BorderLayoutDemo extends JFrame
12     implements ActionListener {
13
14     private JButton buttons[];
15     private String names[] = { "Hide North", "Hide South",
16        "Hide East", "Hide West", "Hide Center" };
```

```
17        private BorderLayout layout;
18
19        // set up GUI and event handling
20        public BorderLayoutDemo()
21        {
22           super( "BorderLayout Demo" );
23
24           layout = new BorderLayout( 5, 5 );
25
26           // get content pane and set its layout
27           Container container = getContentPane();
28           container.setLayout( layout );
29
30           // instantiate button objects
31           buttons = new JButton[ names.length ];
32
33           for ( int count = 0; count < names.length; count++ ) {

34              buttons[ count ] = new JButton( names[ count ] );
35              buttons[ count ].addActionListener( this );
36           }
37
38           // place buttons in BorderLayout; order not important
39           container.add( buttons[ 0 ], BorderLayout.NORTH );
40           container.add( buttons[ 1 ], BorderLayout.SOUTH );
41           container.add( buttons[ 2 ], BorderLayout.EAST );
42           container.add( buttons[ 3 ], BorderLayout.WEST );
43           container.add( buttons[ 4 ], BorderLayout.CENTER );
44
45           setSize( 300, 200 );
46           setVisible( true );
47        }
48
49        // handle button events
50        public void actionPerformed( ActionEvent event )
51        {
52           for ( int count = 0; count < buttons.length; count++ )
53
54              if ( event.getSource() == buttons[ count ] )
55                 buttons[ count ].setVisible( false );
56              else
57                 buttons[ count ].setVisible( true );
58
59           // re-layout the content pane
60           layout.layoutContainer( getContentPane() );
61        }
62
63        // execute application
64        public static void main( String args[] )
```

```
65        {
66           BorderLayoutDemo application = new BorderLayoutDemo();
67
68           application.setDefaultCloseOperation(
69              JFrame.EXIT_ON_CLOSE );
70        }
71
72     }  // end class BorderLayoutDemo
```



مدير العرض GridLayout

يقوم مدير العرض هذا بتقسيم الحاوية container على شكل شبكه Grid بحيث نقوم بوضع العناصر في صفوف وأعمدة، كل خلية في الشبكة لها نفس الطول والعرض، ويتم وضع العناصر في الشبكة ابتداء من الخلية الواقعة أعلى الشبكة من اليسار وتستمر عملية الاضافة من اليسار لليمين حتى يمتلئ الصف ثم ننتقل للصف الذي يليه. المثال التالي يوضح استخدام GridLayout.

```
1
2     // Demonstrating GridLayout.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
```

```
 7
 8    // Java extension packages
 9    import javax.swing.*;
10
11    public class GridLayoutDemo extends JFrame
12       implements ActionListener {
13
14       private JButton buttons[];
15       private String names[] =
16          { "one", "two", "three", "four", "five", "six" };
17       private boolean toggle = true;
18       private Container container;
19       private GridLayout grid1, grid2;
20
21       // set up GUI
22       public GridLayoutDemo()
23       {
24          super( "GridLayout Demo" );
25
26          // set up layouts
27          grid1 = new GridLayout( 2, 3, 5, 5 );
28          grid2 = new GridLayout( 3, 2 );
29
30          // get content pane and set its layout
31          container = getContentPane();
32          container.setLayout( grid1 );
33
34          // create and add buttons
35          buttons = new JButton[ names.length ];
36
37          for( int count = 0; count < names.length; count++ ) {
38             buttons[ count ] = new JButton( names[ count ] );
39             buttons[ count ].addActionListener( this );
40             container.add( buttons[ count ] );
41          }
42
43          setSize( 300, 150 );
44          setVisible( true );
45       }
46
47       // handle button events by toggling between layouts
48       public void actionPerformed( ActionEvent event )
49       {
50          if ( toggle )
51             container.setLayout( grid2 );
52          else
53             container.setLayout( grid1 );
54
55          toggle = !toggle;   // set toggle to opposite value
56          container.validate();
57       }
58
59       // execute application
60       public static void main( String args[] )
61       {
```

```
62          GridLayoutDemo application = new GridLayoutDemo();
63
64          application.setDefaultCloseOperation(
65             JFrame.EXIT_ON_CLOSE );
66       }
67
68    }  // end class GridLayoutDemo
```

تمارين

س١) حدد الأخطاء في كل من الجمل التالية:

```
a) buttonName =JButton("Caption");
b) JLabel aLabel, JLabel;
c) TextField = new JTextField(50, "Default Text");
d) Container c = getContentPane();
   setLayout (new BorderLayout());
   button1 = newJButton ("North Star");
   button2 = newJButton ("South Pole");
   c.add(button1);
   c.add(button2);
```

س٢) قم برسم الشكل التالي من غير اضافة أي عمليات على الرسم

س٣) قم برسم شكل الآلة الحاسبة التالية، واكتب برنامج جافا ليقوم باجراء العملية التي يختارها المستخدم وطباعة الناتج في صندوق النص أعلى الشكل.

(استخدم العناصر الرسومية JButtons، JTextField في الحل)

7	8	9	/
4	5	6	*
1	2	3	-
0	.	=	+

برمجة ٣

الروتين

الجدارة:

أن يكون المتدرب قادراً على التمييز بين أنواع الملفات المختلفة، وكتابة برنامج جافا و يستطيع التعامل مع الملفات..

الأهـداف:

بنهاية هذه الوحدة، عليك أن تكون قادرا على كتابة برنامج جافا يقوم بما يلي:

١. تعريف كائن يحتوي على الملف المراد معالجته
٢. فتح الملف
٣. القراءة من الملف
٤. الكتابة على الملف
٥. تحديد نهاية الملف

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- دفتر
- قلم

معالجة الملفات.

مقدمة

إن تخزين الملفات في متغيرات ومصفوفات هو تخزين مؤقت لها، حيث إن البيانات تفقد عند انتهاء فترة حياة المتغير variable scope أو عند انتهاء تنفيذ البرنامج، لذلك تستخدم البرامج الملفات كوسيلة لتخزين البيانات لفترات زمنية طويلة تتعدى مرحلة تنفيذ البرنامج. يقوم الحاسب بتخزين الملفات في الذاكرة الثانوية مثل القرص الصلب، الأشرطة الممغنطة وغيرها.

في هذه الوحدة سنتعرف على كيفية القراءة من الملفات و الكتابة عليها، ومعالجة البيانات المخزنة على شكل ملفات.

تعتبر عملية معالجة الملفات واحدة من أهم الإمكانيات الواجب توفرها في لغة البرمجة التي ستستخدم لبناء التطبيقات التجارية التي تتعامل مع كمية كبيرة من البيانات، مثل نظام دفع الرواتب.

تنقسم الملفات حسب طريقة الوصول للبيانات فيها إلى:

a. ملفات الوصول التتابعي Sequential Access Files

في هذا النوع من الملفات للوصول إلى سجل معين في الملف عليك قراءة الملف من البداية سجل يليه آخر حتى تصل إلى السجل المطلوب، وكذلك عند الكتابة على الملف

b. ملفات الوصول العشوائي Random Access Files

في هذا النوع من الملفات يمكنك الوصول إلى السجل المطلوب مباشرة دون الحاجة للمرور على كل السجلات التي قبله.

كما ويمكن تقسيم الملفات حسب نوع البيانات إلى:

i. ملفات نصية Text files : حيث تتم قراءة بيانات الملف والكتابة عليه على شكل حروف Characters

ii. ملفات ثنائية Binary files : وهنا يتم التعامل مع بيانات الملف على أنها مجموعة من البايت Bytes

ولكل نوع من الملفات تطبيقات معينة يمكن استخدامه فيها، وستقتصر دراستنا في هذه الوحدة على ملفات الوصول التتابعي والملفات النصية.

القراءة من ملف

إليك الآن المثال الأول، والذي يوضح كيفية تعريف الملفات وفتحها للقراءة منها سطراً سطراً.

```
1    import java.io.*;
2    //Class Definition
3    class ReadTextFilel
4    {
5       public static void main (String args[]) throws IOException
6       {
7           String fileName = “c:/temp/toRead.txt”;
8           String line;
9            BufferedReader in = new BufferedReader (new
10                                                   FileReader (filename) );
11           line = in.readLine();
12           while (line != null) // continue until end of file;
13           {
14               System.out.prinln(line);
15                Line = in.readLine();
16           }
17           in.close();
18       }
19   }
20
```

السطر الأول: يقوم السطر الأول باستيراد المجموعة package المسماة java.io وتحتوي علىجميع الفصائل الخاصة بالملفات ومعالجتها.

السطر ٢ -٦ : في هذه الأسطر قمنا بتعريف اسم الفصيل ReadTextFile1 والطريقة الرئيسة main، لاحظ أن استخدامنا للملفات قد يتسبب في حدوث بعض الاسثناءات مثل "الملف غير موجود FileNotFound" وغيره لذلك وجب علينا تحديد أن الطريقة الرئيسة main قد تطلق استثناء، وذلك باستخدام الجملة throws IOException.

السطر ٧: لفتح واستخدام ملف معين لابد لنا من تعريف اسم الملف وموقع تخزين الملف وذلك بتحديد المسار الخاص به، وقد قمنا بذلك من خلال تعريف متغير نصي اسمه fileName يمثل اسم الملف والمسار الخاص به.

```
String fileName = “c:/temp/toRead.txt”;
```

السطر ٨: كما قمنا بتعريف متغير نصي آخر هو line والذي سنستخدمه لتخزين السطر المقروء من الملف من أجل طباعته على الشاشة.

السطر ٩ -١٠: في هذا السطر نقوم بتعريف كائن اسمه in من نوع BufferedReader حيث إن الكائن سيستخدم في قراءة ملف هو fileName معرف على شكل كائن للقراءة من نوع FileReader.

```
BufferedReader in = new BufferedReader  (new FileReader (fileName) );
```

السطر ١٢: بعد ان قمنا بتعريف الكائن in والذي سنقرأ من خلاله البيانات من الملف على شكل سطر يليه سطر. نقوم الآن بقراءة السطر الأول

```
line = in.readln ();
```

في هذه الجمله نطلب قراءة سطر من الكائن in، ثم تخزين السطر في المتغير line والانتقال إلى السطر التالي.

السطر ١١: للاستمرار في قراءة البيانات من الملف وحيث إن عدد الأسطر فيه غير محدد فإننا سنستخدم التكرار (طالما while) لتحديد متى ينتهي الملف حيث إن الملف ينتهي عندما لا يعود هناك أسطر للقراءة أي عند قراءة null.

```
while (line != null)
```

السطر ١٢ -١٥: نقوم هنا بطباعة السطر المقروء على الشاشة، والانتقال لقراءة السطر الذي يليه، وهكذا حتى ننتهي من قراءة وطباعة الملف كله.

```
 {
      System.out.prinln(line);
      line = in.readLine();
}
```

السطر ١٦: بعد فتح الملف والانتهاء من قراءة البيانات منه، نقوم الآن بإغلاق الملف وإنهاء عمل الكائن in.

```
in.close();
```

بعد أن تعرفنا على كيفية قراءة الملف سطر يليه سطر، سنتعرف الآن على كيفية القراءة من الملف ولكن كلمة تليها كلمة، حيث إن هناك العديد من التطبيقات التي نحتاج فيها إلى قراءة كلمات من الملف.

```
1     import java.io.*;
2     class ReadWithTokenizer
3     {
4        public static void main (String args[]) throws IOException
5        {
6
7            String fileName = "c:/temp/toRead.txt";
8            BufferedReader in = new BufferedReader  (
9                                        new FileReader (fileName) );
10            StreamTokenizer reader = new StreamTokenizer(in);
11            reader.nextToken();
12           While (reader.ttype != StreamTokenizer.TT_EOF)
13             //continue until end of file
14             {
15                  String word = reader.sval;
16                  System.out.println (word);
17                  Reader.nextToken();
18             }
19             in.close();
20        }
21    }
22
23
```

السطر الأول: يقوم السطر الأول باستيراد المجموعة package المسماة java.io وتحتوي علىجميع الفصائل الخاصة بالملفات ومعالجتها.

السطر ٢ -٦ : في هذه الأسطر قمنا بتعريف اسم الفصيلة ReadwithTokenizer والطريقة الرئيسة main.

السطر ٧: نقوم هنا بتعريف اسم الملف وموقع تخزينه، وقد قمنا بذلك من خلال تعريف متغير نصي اسمه fileName يمثل اسم الملف والمسار الخاص به.

```
String fileName = "c:/temp/toRead.txt";
```

السطر ٨ -٩: في هذا السطر نقوم بتعريف كائن اسمه in من نوع BufferedReader حيث إن الكائن سيستخدم في قراءة ملف هو fileName معرف على شكل كائن للقراءة من نوع FileReader.

```
BufferedReader in = new BufferedReader (new FileReader (fileName) );
```

السطر ١٠: حيث إننا نريد قراءة الملف على شكل كلمات فإننا سنتعامل مع الملف على أنه مجموعة من الكلمات Tokens لذلك سنقوم بإدخال الكائن الذي يمثل ملف القراءة كباراميتر لمنشئ كائن جديد من نوع StreamTokenizer هو reader، الذي سيمكننا من القراءة بشكل كلمات

```
StreamTokenizer reader = new StreamTokenizer(in);
```

السطر ١١: لقراءة الكلمة الأولى في الملف سنستخدم الطريقة المسماة nextToken() والخاصة بالفصيل StreamTokenizer، وستتم منادات الطريقة من خلال الكائن reader.

```
reader.nextToken();
```

السطر ١٢: سنستمر في قراءة الكلمات من الملف حتى نصل إلى نهايته، حيث إن كل ملف له حرف يمثل نهايته، هذا الحرف مخفي في لغة جافا abstracted، ويمكننا الرجوع له من خلال المعرف الثابت TT_EOF الموجود داخل الفصيل StreamTokenizer، وبناء على ذلك فإن جملة التكرار في البرنامج هي:

```
while (reader.ttype != StreamTokenizer.TT_EOF)
```

السطر ١٥: عند استخدام الطريقة reader.nextToken فإن الكلمة التالية في الملف ستنتقل إلى الكائن reader، هذه الكلمة إما أن يكون لها قيمة حرفيه أو قيمة رقمية لذلك سنقوم بقراءة هذه القيمة وتخزينها في متغير من نوع حرفي أو رقمي باستخدام الطريقة nval أو sval على التوالي.

```
String word = reader.sval;
```

السطر ١٦: سنقوم بطباعة الكلمة من خلال الطريقة القياسية

```
System.out.println (word);
```

السطر ١٧:قبل نهاية التكرار لا بد لنا من قراءة الكلمة التاليه حتى نتمكن من التحقق من الوصول إلى نهاية الملف أو طباعة الكلمة وهكذا.

```
reader.nextToken ();
```

السطر ١٩: بعد الانتهاء من قراءة البيانات من الملف سنقوم باغلاقه من خلال الجملة التالية:

```
in.close ();
```

كما يمكنك التاكد من نوع الكلمة المقروءة من الملف وتخزينها في المتغير المناسب، لاحظ الجمل التالية:

```
While (reader.nextToken() != StreamTokenizer.TT_EOF)
//continue until end of file
{
    if (reader.ttype == StreamTokenizer.TT_WORD)
          System.out.println(" A word: " + reader.sval);
    Else if (reader.ttype == StreamTokenizer.TT_NUMBER)
           System.out.println(" A number: " + reader.nal);
}
```

لقد استخدمنا في المثال أعلاه مجموعة من المعرفات الثابتة الموجودة داخل الفصيل StreamTokenizer مثل TT_WORD ويمثل هذا المعرف القيم الحرفية String، في حين أن المعرف TT_NUMBER يمثل القيم العددية. كما أن لكل كلمة يتم قراءتها من الملف إلى الكائن reader لها نوع يحدد بالمتغير ttype. وبالتالي نستطيع تكوين جملة شرطية كما يلي:

```
if (reader.ttype == StreamTokenizer.TT_WORD)
```

الكتابة على ملف

لقد رأينا في الأمثلة أعلاه كيف نقرأ من ملف وكيف نتعامل مع البيانات على شكل كلمات أو أسطر، وسوف نتعلم الآن كيف نكتب على ملف. انظر إلى البرنامج التالي:

```
1  import java.io.*;
2  public class WriteTextFile
3  {
5      public static void main (String args[] throws IOExcepetion
6      {
7          String filename = “reaper.txt”
8           PrintWriter print = new PrintWriter( new BufferedWriter (
9                                    new FileWriter (filename)));
10          print.println(“College of Telecommunication and Information”);
11          print.println(“Computer Department”);
12          print.println(“Programming”);
13          print.println(“Java 3”);
14          print.close();
15      }
16 }
```

في السطر ٧ قمنا بتحديد اسم الملف المراد الكتابة عليه، ثم قمنا في السطر الثامن بتعريف كائن اسمه printمن نوع PrintWriter وهو الكائن الذي سنستخدمه للكتابة على الملف.

إن عملية الكتابة على الملف تتم من خلال استخدام الطريقة print أو println مسبوقة باسم الكائن الذي يمثل الملف الخاص بالكتابة.

اخيرا سنعرض لك مثالاً يوضح كيفية مناداة طريقه معرفة من قبل المستخدم لتحميل البيانات من ملف إلى مصفوفة. وطريقه أخرى لتخزين البيانات ونقلها من المصفوفة إلى الملف.

```
1
2    import java.io.*
3    class ReadWrite
4    {
5        public static void main(String[] args) throws IOException
6        {
7             String[] line = new String[10];
8             load (line);
9
10          /* -------- نكتب هنا مجموعة من العمليات المختلفهm ---------------- */
11
12          commit (line);
13       }
14      □ سنعرف الان طريقه لتحميل البيانات من الملف إلى مصفوفة من الكائنات //
15
16      Public static void load (String[] line) throws IOException
17      {
18         String filename =”c:/temp/toRead.txt”;
19         BufferedReader in = new BufferedReader (new
20                                                    FileReader(filename));
21         Line[0] = in.readln();
22         int i = 0;
23         while (line[i] != null) // استمر حتى نهاية الملف
24         {
25            System.out.println(line[i]);
26             i++;
27             Line[i] = in.readln();
28         }
29         in.close();
30      }
```

```
31  // سنعرف الآن طريقه لحفظ البيانات التي في المصفوفة إلى الملف
32
33     public static void commit (String[] line) throws IOException
34     {
35        String filename =”c:/temp/toRead.txt”;
36        BufferedWriter print = new BufferedWriter (new
37
38  FileWriter(filename));
39        int i = 0;
40        while (line[i] != null) // استمر حتى نهاية الملف□
41        {
42          print.println(line[i]);
43            i++;
44        }
45       print.close();
46     }
47  }
48
```

تمارين

س١) لماذا نحتاج استخدام الملفات خصوصا في البرامج التي تتعامل مع كم كبير من البيانات؟

____

____

____

س٢) تنقسم الملفات حسب طريقة الوصول للبيانات إلى قسمين هما:

١. ______________________________

٢. ______________________________

س٣)) تنقسم الملفات حسب طريقة التعامل مع البيانات إلى قسمين هما:

١. ______________________________

٢. ______________________________

س٤) لديك شركه تحتاج فيها إلى متابعة بيانات الموظفين وبيانات الأقسام وتحديد القسم الذي يعمل فيه كل موظف. قم بكتابة فصيلين تنشئ من خلالهما مجموعة من الأنواع (ADT) لكل من الموظفين والأقسام، ثم اكتب برنامجاً يتعامل مع هذين الفصلين، ويقوم بتخزين البيانات على ملفات وقراءتها من الملفات مرة أخرى.



برمجة ٣

الاتصال بقواعد البيانات

الجدارة:

أن يكون المتدرب قادراً على تعريف مشغل لقاعدة بيانات معينة وكتابة برنامج جافا يقوم بالتعامل مع قاعدة البيانات من خلال هذا المشغل.

الأهــداف:

بنهاية هذه الوحدة، عليك أن تكون قادرا على:

١. إنشاء مشغل بقاعدة البيانات
٢. كتابة برنامج جافا يقوم بالتعامل مع قاعدة البيانات من خلال المشغل

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- دفتر
- قلم

اتصال جافا بقواعد البيانات

مقدمه

توفر لغة الجافا كمثيلاتها من لغات البرمجة عدة طرق لتخزين المعلومات على ملفات مخزنة في الذاكرة الثانوية مثل: الملفات التسلسلية Sequential files وملفات الوصول العشوائي Random-access files وعلى الرغم من فائدة هاتين الطريقتين في حفظ البيانات، إلا أنهما لا تمتلكان الإمكانيات الكافية للاستعلام عن البيانات بشكل مناسب. إن أنظمة قواعد البيانات لا توفر فقط القدرة على معالجة الملفات،ولكنها تنظم البيانات بطريقة تسمح لها تنفيذ عمليات استعلام معقده. إن أشهر أنواع قواعد البيانات المستخدمة حاليا هي قواعد البيانات العلائقية Relational Database systems وفي هذا النوع من قواعد البيانات تستخدم لغه تسمى لغة الاستعلام الهيكلية Structured Query Language (SQL) والتي تستخدم للاستعلام عن البيانات التي تحقق شرطاً معيناً في قاعدة البيانات. وللاستفادة من قواعد البيانات هذه فإن معظم لغات البرمجة ومنها جافا توفر إمكانية عمل اتصال بقواعد البيانات المختلفة وكتابة جمل SQL للاستعلام عن البيانات. ومن أشهر قواعد البيانات المستخدمه ما يلي: Microsoft Access, Sybase, Oracle, Informix, Microsoft SQL Server وغيرها. في هذه الوحدة سنتعرف على كيفية كتابة برنامج يتصل بقاعدة بيانات Microsoft Access و Oracle، آخذين بعين الاعتبار إن لدى القارئ الماماً بلغة SQL ومفاهيم قاعدة البيانات.

قاعدة البيانات المستخدمة

سنقوم الآن بوصف قاعدة البيانات المستخدمة في هذه الوحدة واسمها الكتب Books، والتي يمكن انشاؤها كقاعدة بيانات Access أو Oracle أو غيرهما.

تتكون قاعدة البيانات هذه من أربعة جدأول هي: المؤلف، الناشر، ISBN للمؤلف وجدول العنوان، وتفصيلاتها كما يلي:

جدول المؤلف	
الحقل	الوصف
رقم المؤلف AutherID	رقم المؤلف في قاعدة البيانات(المفتاح الرئيس Primary Key)
الاسم الأول FirstName	الاسم الأول للمؤلف
الاسم الأخير LastName	الاسم الأخير للمؤلف
سنة الميلاد YearBorn	السنة التي ولد فيها المؤلف

وإليك عينة من البيانات الموجودة في جدول المؤلف:

AutherId	FirstName	LastName	YearBorn
1	Ali	Suliman	1960
2	Salem	Khalid	1975
3	Abdullah	Amer	1963

جدول الناشر	
الحقل	الوصف
رقم الناشر PublisherID	رقم الناشر في قاعدة البيانات (مفتاح رئيس)
اسم الناشر PublisherName	اسم الناشر

عينة من البيانات الموجودة في جدول الناشر:

PublisherID	PublisherName
1	Prentice Hall
2	Prentice Hall PTR

جدول **ISBN**المؤلف	
الحقل	الوصف
ISBN	رقم ISBN للكتاب
رقم المؤلف AutherID	رقم المؤلف صاحب الكتاب

عينة من البيانات الموجودة في جدول ISBNالمؤلف:

ISBN	AutherID
0-13-010671-2	1
0-13-015231-2	2
0-14-044131-7	1
0-11-028271-4	3
0-10-070471-1	2

جدول العنوان	
الحقل	الوصف
ISBN	رقم ISBN للكتاب
العنوان Title	عنون الكتاب
رقم الطبعة EditionNo	رقم طبعة الكتاب
تاريخ النشر YearPublished	سنة نشر الكتاب
رقم الناشر PublisherId	رقم ناشر الكتاب

عينة من البيانات الموجودة في جدول العنوان:

ISBN	Title	Edition No	Year-Published	PublisherID
0-13-010671-2	C How to program	2	1994	1
0-13-015231-2	C++ How to Program	3	1997	1
0-14-044131-7	Java How to program	2	1992	1
0-11-028271-4	Oracle PL/SQL	2	1999	2
0-10-070471-1	Internet Programming	1	1998	1

الخطوة اللاحقة هي معرفة كيفية تعريف قاعدة البيانات أعلاه بلغة الجافا والتي يمكنك إنشاؤها بأحد أنظمة معالجة قواعد البيانات العلائقية Relational database Management Systems مثل Access ، Oracle أو غيرها.

مثال: استرجع بيانات جدول المؤلفين مرتبة حسب الاسم الأول.

```
SELECT * from Author
ORDER BY FirstName
```

البرنامج الأول:

في هذا المثال سنقوم بإجراء عمليات استعلام بسيطة على قاعدة بيانات الكتب واسترجاع البيانات عن كل المؤلفين وعرضها على وحدة JTable. يوضح البرنامج أدناه كيفية الاتصال بقاعدة البيانات، الاستعلام من قاعدة البيانات وعرض النتائج.

```
1    // Fig. 18.24: TableDisplay.java
2    // This program displays the contents of the Authors table
3    // in the Books database.
4    import java.sql.*;
5    import javax.swing.*;
6    import java.awt.*;
7    import java.awt.event.*;
8    import java.util.*;
9
10   public class TableDisplay extends JFrame {
11     private Connection connection;
12     private JTable table;
13
14     public TableDisplay()
15     {
16       // The URL specifying the Books database to which
17       // this program connects using JDBC to connect to a
18       // Microsoft ODBC database.
19       String url = "jdbc:odbc:Books";
20       String username = "anonymous";
21       String password = "guest";
22
23       // Load the driver to allow connection to the database
24       try {
25         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
26
27         connection = DriverManager.getConnection(
```

```
28                url, username, password );
29            }
30            catch ( ClassNotFoundException cnfex ) {
31
32              System.err.println(
33                "Failed to load JDBC/ODBC driver." );
34              cnfex.printStackTrace();
35              System.exit( 1 );  // terminate program
36            }
37            catch ( SQLException sqlex ) {
38              System.err.println( "Unable to connect" );
39              sqlex.printStackTrace();
40            }
41
42            getTable();
43
44            setSize( 450, 150 );
45            show();
46          }
47
48          private void getTable()
49          {
50            Statement statement;
51            ResultSet resultSet;
52
53            try {
54              String query = "SELECT * FROM Author";
55
56              statement = connection.createStatement();
57              resultSet = statement.executeQuery( query );
58              displayResultSet( resultSet );
59              statement.close();
60            }
61            catch ( SQLException sqlex ) {
62              sqlex.printStackTrace();
63            }
64          }
65
66          private void displayResultSet( ResultSet rs )
67            throws SQLException
68          {
69            // position to first record
```

```
70          boolean moreRecords = rs.next();
71
72          // If there are no records, display a message
73          if ( ! moreRecords ) {
74            JOptionPane.showMessageDialog( this,
75              "ResultSet contained no records" );
76            setTitle( "No records to display" );
77            return;
78          }
79
80          setTitle( "Authors table from Books" );
81
82          Vector columnHeads = new Vector();
83          Vector rows = new Vector();
84
85          try {
86            // get column heads
87            ResultSetMetaData rsmd = rs.getMetaData();
88
89            for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
90              columnHeads.addElement( rsmd.getColumnName( i ) );
91
92            // get row data
93            do {
94              rows.addElement( getNextRow( rs, rsmd ) );
95            } while ( rs.next() );
96
97            // display table with ResultSet contents
98            table = new JTable( rows, columnHeads );
99            JScrollPane scroller = new JScrollPane( table );
100           getContentPane().add(
101             scroller, BorderLayout.CENTER );
102           validate();
103         }
104         catch ( SQLException sqlex ) {
105           sqlex.printStackTrace();
106         }
107       }
108
109       private Vector getNextRow( ResultSet rs,
110                                  ResultSetMetaData rsmd )
111          throws SQLException
```

```
112      {
113        Vector currentRow = new Vector();
114
115        for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
116          switch( rsmd.getColumnType( i ) ) {
117            case Types.VARCHAR:
118                currentRow.addElement( rs.getString( i ) );
119              break;
120            case Types.INTEGER:
121                currentRow.addElement(
122                  new Long( rs.getLong( i ) ) );
123              break;
124            default:
125              System.out.println( "Type was: " +
126                rsmd.getColumnTypeName( i ) );
127          }
128
129        return currentRow;
130      }
131
132      public void shutDown()
133      {
134        try {
135          connection.close();
136        }
137        catch ( SQLException sqlex ) {
138          System.err.println( "Unable to disconnect" );
139          sqlex.printStackTrace();
140        }
141      }
142
143      public static void main( String args[] )
144      {
145        final TableDisplay app = new TableDisplay();
146
147        app.addWindowListener(
148          new WindowAdapter() {
149            public void windowClosing( WindowEvent e )
150            {
151              app.shutDown();
152              System.exit( 0 );
153            }
```

```
154           }
155         );
156      }
       }
```

لاحظ جملة

import java.sql.*

تقوم هذه الجملة باستيراد الحزمة java.sql والتي تحتوي على كل الفصائل Classes المتعلقة بإدارة قواعد البيانات العلائقية في لغة الجافا. في حين أن جملة

private Connection connection;

تعرف مرجعية اتصال، ويتم تعريف كائن اتصال Connection Object لإدارة الاتصال بين برنامج جافا وبين قاعدة البيانات، كما أنه يوفر إمكانية تنفيذ جملSQL لمعالجة قاعدة البيانات والحركات Transactions الموجه إليها.

إن منشئ الفصيلة Class Constructor للفصيله TableDisplay سيقوم بإنشاء الاتصال مع قاعدة البيانات وعند نجاحه بذلك سينفذ الاستعلام المطلوب ويظهر الناتج من خلال مناداة الدالة getTable.

```
String url = "jdbc:odbc:Books";
String username = "anonymous";
String password = "guest";
```

حتى يتمكن منشئ الفصيلة من إنشاء الاتصال، لا بد من تحديد ثلاثة أمور موضحة في الجمل الثلاث أعلاه وهي: موقع قاعدة البيانات المراد الاتصال بها من خلال تحديد عنوان URL والذي يحدد البروتوكول الرئيس jdbc والبروتوكول الفرعي odbc المستخدمين في تحقيق الاتصال يليهما بيان اسم قاعدة البيانات. كما نحتاج لتحديد اسم المستخدم username وكلمة المرور password التي سيتم من خلالها الاتصال بقاعدة البيانات حيث إننا عند تعريف مصدر البيانات كما سيتضح في الموضوع التالي قد حددنا ضرورة إدخال اسم المستخدم وكلمة المرور .

وليتمكن أي برنامج جافا من الوصول إلى أي قاعدة بيانات بتقنية ODBC فإن لغة جافا توفر مشغل Driver لتعريف كيفية اتمام الاتصال اسمه jdbc.odbc.jdbcodbcDriver ويجب تحميل هذا المشغل قبل إجراء الاتصال مع قاعدة البيانات.

```
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
```

إن الجملة أعلاه تستخدم الدالة forName لتحميل تعريف الفصيلة class التي تحتوي على مشغل قاعدة البيانات Database driver، وتحميل هذه الفصيلة قد يتسبب في استثناء في حالة أن الفصيلة المطلوبة لا

يمكن تحميلها وهذا الاستثناء هو java.lang.classNotFound لذلك فقد وضعت في جزء try block لنتمكن من متابعة الاستثناء ومعالجته في جزء catch block.

```
connection = DriverManager.getConnection(url, username, password );
```

في الجملة أعلاه نستخدم الدالة getConnection التابعة للفصيلة DriverManager من أجل الاتصال بقاعدة البيانات المحددة بالمتغير URL وقد تم تحديد اسم المستخدم وكلمة المرور أيضا. وفي حالة عدم القدرة على الاتصال بقاعدة البيانات سيحدث استثناء هو java.sql.sqlException أما في حالة الاتصال بقاعدة البيانات فإنه سيتم مناداة الدالة getTable لاسترجاع البيانات.

تقوم الدالة gettable بالاستعلام من قاعدة البيانات ومن ثم مناداة الدالة displayResultSet لإنشاء كائن رسومي JTable لإظهار الناتج من خلاله.

```
Statement statement
```

السطر أعلاه يعرف مرجع لجملةSQL من نوع Statement (هذا النوع موجود ضمن الحزمة java.sql) وهذا المرجع سيستخدم للرجوع إلى الكائن الذي سيتم حفظ جملة SQL بداخله لنقلها إلى قاعدة البيانات لتنفيذها.

```
ResultSet resultSet
```

في هذا السطر قمنا بتعريف الكائن resultSet والذي سيتم إرجاعه إلى برنامج الجافا من قاعدة البيانات، وبداخله ناتج تنفيذ جملة SQL.

```
statement = connection.createStatement();
```

تقوم الجملة أعلاه بمناداة الدالة CreateStatement لايجاد كائن statement والذي سيستخدم للاستعلام من قاعدة البيانات

```
resultSet = statement.executeQuery(query);
```

تقوم هذه الجملة بعمل الاستعلام من خلال مناداة الدالة executeQuery، هذه الدالة ستعيد كائن من قاعدة البياتات يحتوي على ناتج تنفيذ الاستعلام. الكائن resultSet سيمرر إلى الدالة displayResult وبعدها يتم إغلاق الجملة للدلالة على الانتهاء من معالجة الجملة.

السطر ٦٩ من الدالة displayResultSet

```
boolean moreRecords = rs.next();
```

بعد تنفيذ هذا السطر فإن المؤشر الممثل بالمتغيرmoreRecords يشير إلى السجل الأول في النتائج الموجودة في الكائن ResultSet وذلك باستخدام الدالة next والتي تحرك المؤشر إلى السجل التالي حيث إنه يشير مبدئيا إلى ما قبل السجل الأول. وتلاحظ أن الدالة next ترجع قيمة بووليه Boolean تبين من خلالها فيما إذا كان باستطاعتها الانتقال إلى السجل التالي إن وجد (True)، أو عدم وجود تالي وبالتالي يكون

الناتج (False). في حالة أنه كان هناك تالي فإن السطر ٨١ يعرف مصفوفة Vector لتخزين أسماء الاعمدة الموجودة في الناتج ResultSet والسطر ٨٢ يعرف مصفوفة لتخزين سجلات البيانات من الكائن ResultSet ، هذه المصفوفات ستسخدم مع منشئ JTable لبناء جدول يظهر البيانات من ResultSet.

السطر ٨٦:

```
ResultSetMetaData  rsmd = rs.getMetaData();
```

تقوم هذه الجملة بالحصول على البيانات التفصيلية عن الجدأول الموجودة في ResultSet ،مثل أسماء وأنواع الأعمدة في الجدأول، تسمى هذه البيانات التفصيلية MetaData ومن ثم إسنادها إلى الكائن rsmd. لقد قمنا باستخدام ResultSetMetaData في الأسطر ٨٨ و ٨٩ لاسترجاع اسم كل عمود في الكائن ResultSet وقد تم استخدام الدالة getColumnCount لتحديد عدد الأعمدة والدالة getColumnName لتحديد اسم العمود

السطر ٩٢ لغاية ٩٤:

```
do
      rows.addElement (getNextRow (rs, rsmd ) );
       while (rs.next() );
```

تقوم باسترجاع كل سطر من ResultSet باستخدام الدالة getNextRow ،والمعرفة بالسطر ١٠٨، هذه الدالة لها قيمة مرتجعة من نوع مصفوفه احادية البعد تحتوي البيانات لسطر واحد، لاحظ الشرط rs.next() والذي ينقل المؤشر الخاص بمتابعة الانتقال إلى السجل التالي في الكائن ResultSet، إن وجد، وبالتالي فإن التكرار أعلاه سينتهي عندما لا يبقى هناك سجلات في الكائن ResultSet.

بعد تحويل كل السجلات إلى مصفوفات أحادية البعد، يقوم السطر ٩٧ بإنشاء JTable لإظهار هذه السجلات.

الدالة getNextRow (سطر ١٠٨) تستقبل ResultSet و ResultSetMetaData كباراميترات وتنشئ مصفوفة أحادية تحتوي على سجل واحد من البيانات من ResultSet.

الدالة shutdown في السطر ١٣١ تقوم بإغلاق الاتصال مع قاعدة البيانات باستخدام الدالة close.

تسجيل قاعدة البيانات "الكتب" Books.mdb كمصدر بيانات في مصدر قواعد البيانات المفتوح ODBC.

إن المثال السابق يفترض بأن قاعدة البيانات المستخدمة مسجلة كمصدر بيانات ODBC، وما سنفعله الآن هو التعرف على كيفية تعريف قاعدة بيانات كمصدر بيانات ODBC. للقيام بهذا العمل علينا اتباع الخطوات التالية:

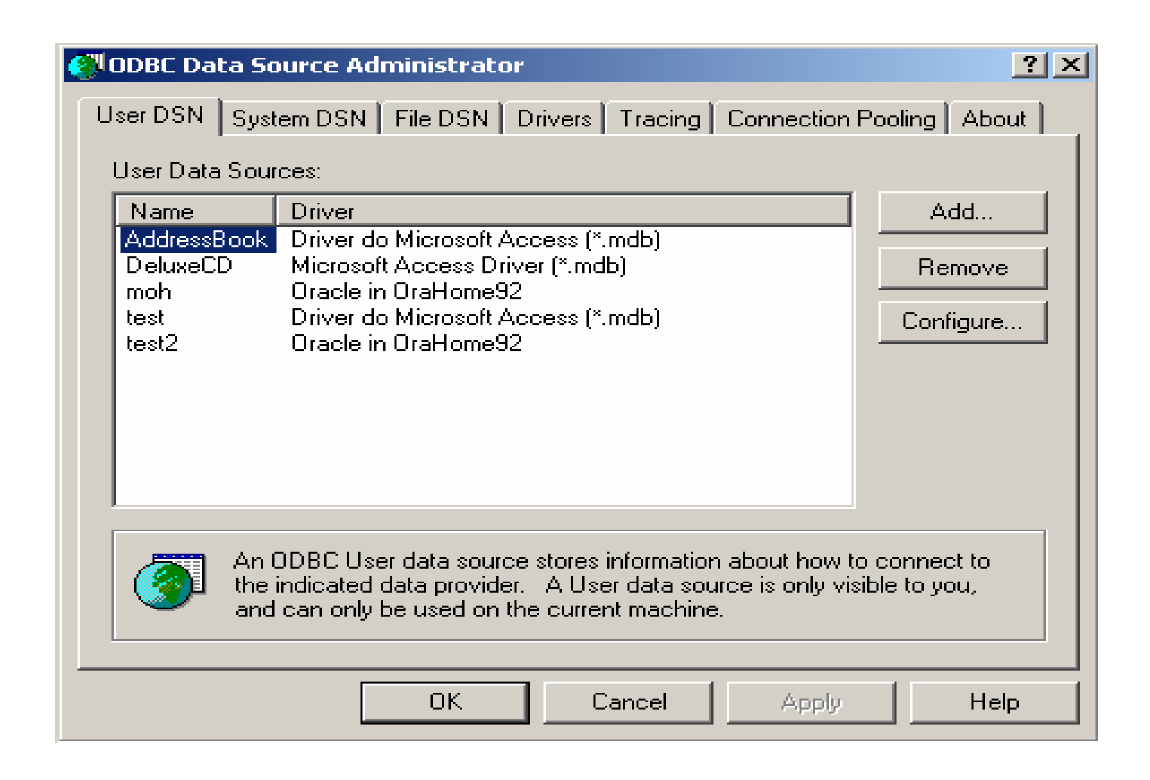
انتقل إلى Control Panel داخل نظام Windows

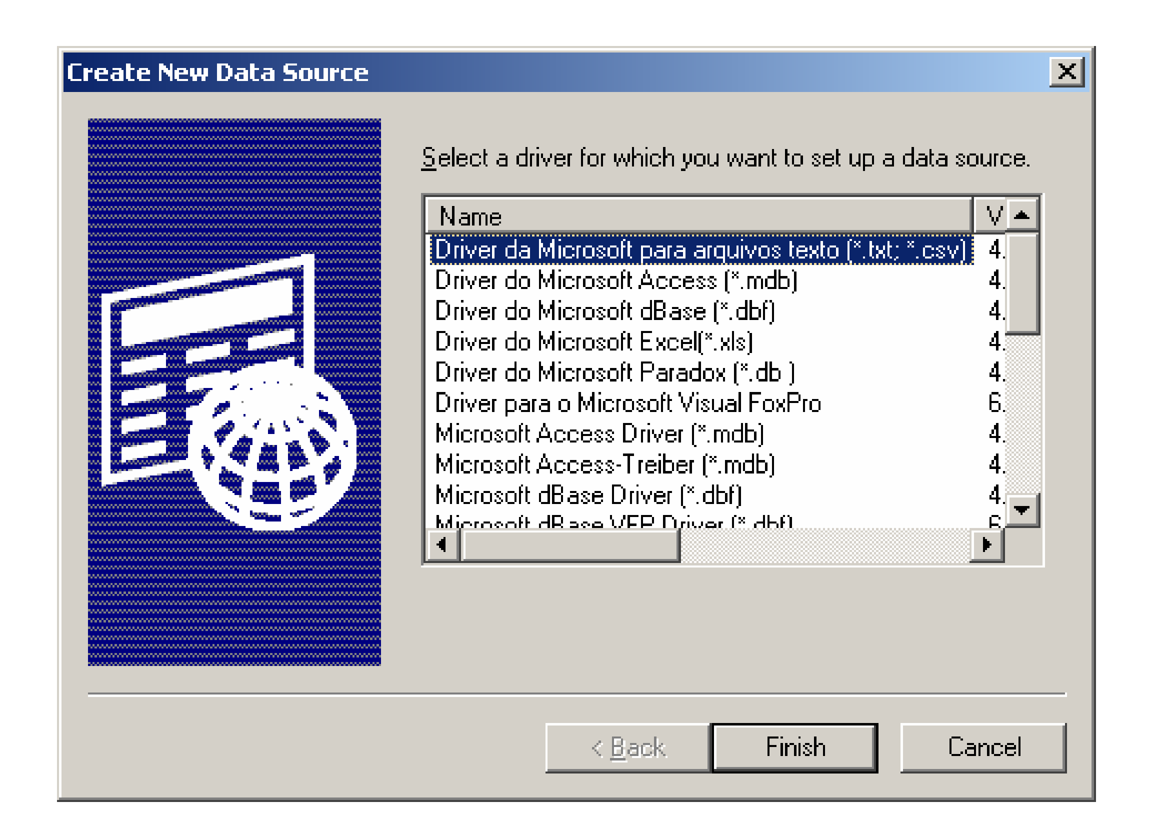
انقر نقرة مزدوجة على الخيار ODBC Data Sources، سيظهر لك الآن صندوق الحوار المبين ادناه.

في قائمة User DSN انقر على الزر Add لإظهار صندوق إنشاء مصدر بيانات جديد كما هو موضح بالشكل٢.

حيث إن قاعدة البيانات الخاصة بنا هي من نوع Access فسوف نستخدم Microsoft Access Driver.

انقر الزر إنهاء.





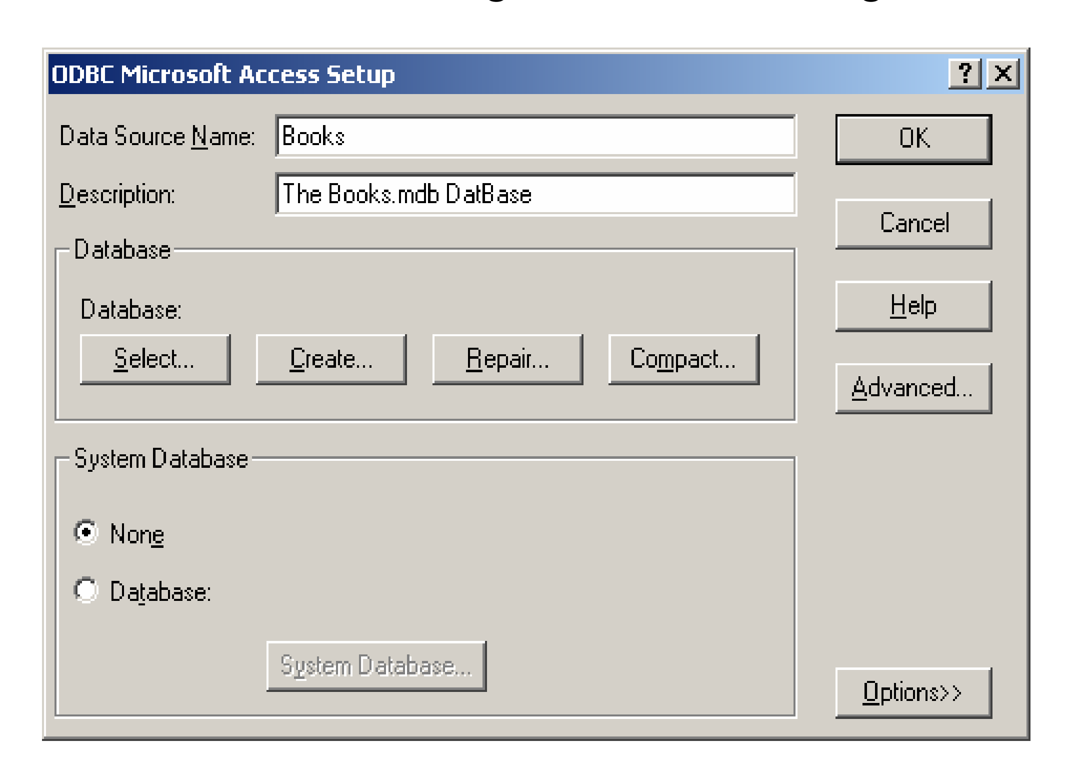
سيظهر لنا الآن صندوق الحوار الخاص بـ ODBC Microsoft Access، حيث سنقوم بتحديد كل مما يلي:

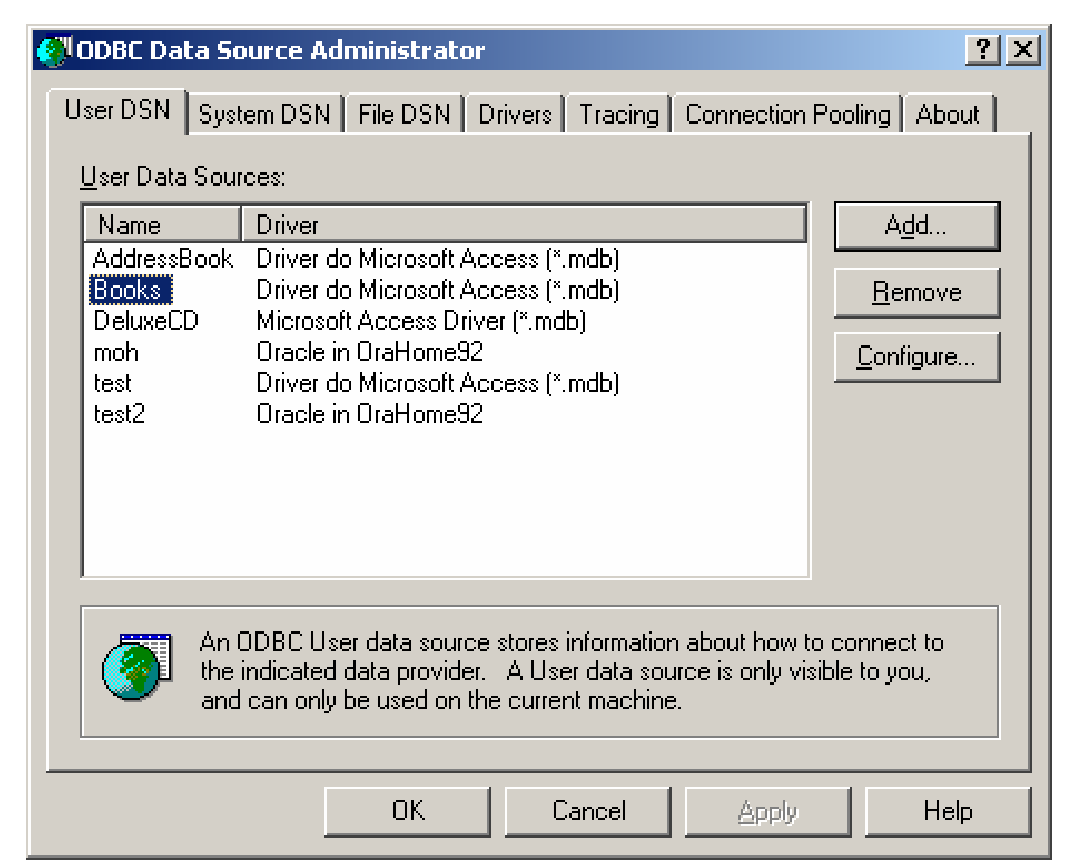
أدخل اسم قاعدة البيانات الذي سيستخدم من قبل JDBC للرجوع لقاعدة البيانات في الحقل Data Source Name.

يمكنك إدخال وصف لقاعدة البيانات (اختياري) في الحقل Description

١. انقر على زر Select
٢. ابحث ثم اختر اسم قاعدة البيانات الخاصة بك (في هذه الحالة نختار Books.mdb)
٣. انقر الزر OK
٤. انقر الزر Advanced لإظهار قائمة الخيارات المتقدمة
٥. أدخل اسم المستخدم anonymous
٦. أدخل كلمة المرور guest
٧. انقر الزر OK للخروج من صندوق الحوار
٨. ١١. انقر الزر OK مرة أخرى للخروج من ODBC Microsoft Access Setup
٩. ١٢. انقر الزر OK مرة أخرى للخروج من ODBC Data Source Administrator

تستطيع الآن تنفيذ البرنامج المكتوب أعلاه لمشاهدة الناتج.





المثال الثاني:

في هذا المثال سنقوم بتعديل المثال الأول بحيث يستطيع المستخدم إدخال أي جملة استعلام، ثم يقوم البرنامج بتنفيذ هذه الجملة في قاعدة البيانات وعرض الناتج على الشاشة.

```
1	// DisplayQueryResults.java
2	// This program displays the ResultSet returned by a
3	// query on the Books database.
4	import java.sql.*;
5	import javax.swing.*;
6	import java.awt.*;
7	import java.awt.event.*;
8	import java.util.*;
9
10	public class DisplayQueryResults extends JFrame {
11	   // java.sql types needed for database processing
12	   private Connection connection;
13	   private Statement statement;
14	   private ResultSet resultSet;
15	   private ResultSetMetaData rsMetaData;
16
```

```
17        // javax.swing types needed for GUI
18        private JTable table;
19        private JTextArea inputQuery;
20        private JButton submitQuery;
21
22        public DisplayQueryResults()
23        {
24          super( "Enter Query. Click Submit to See Results." );
25
26          // The URL specifying the Books database to which
27          // this program connects using JDBC to connect to a
28          // Microsoft ODBC database.
29          String url = "jdbc:odbc:test";
30          String username = "moh";
31          String password = "moh";
32
33          // Load the driver to allow connection to the database
34          try {
35            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
36
37            connection = DriverManager.getConnection(
38              url, username, password );
39          }
40          catch ( ClassNotFoundException cnfex ) {
41            System.err.println(
42              "Failed to load JDBC/ODBC driver." );
43            cnfex.printStackTrace();
44            System.exit( 1 );  // terminate program
45          }
46          catch ( SQLException sqlex ) {
47            System.err.println( "Unable to connect" );
48            sqlex.printStackTrace();
49            System.exit( 1 );  // terminate program
50          }
51
52          // If connected to database, set up GUI
53          inputQuery =
54            new JTextArea( "SELECT * FROM Authors", 4, 30 );
55          submitQuery = new JButton( "Submit query" );
56          submitQuery.addActionListener(
57            new ActionListener() {
58              public void actionPerformed( ActionEvent e )
```

```
59              {
60                 getTable();
61              }
62           }
63        );
64
65        JPanel topPanel = new JPanel();
66        topPanel.setLayout( new BorderLayout() );
67        topPanel.add( new JScrollPane( inputQuery),
68                 BorderLayout.CENTER );
69        topPanel.add( submitQuery, BorderLayout.SOUTH );
70
71        table = new JTable( 4, 4 );
72
73        Container c = getContentPane();
74        c.setLayout( new BorderLayout() );
75        c.add( topPanel, BorderLayout.NORTH );
76        c.add( table, BorderLayout.CENTER );
77
78        getTable();
79
80        setSize( 500, 500 );
81        show();
82     }
83
84     private void getTable()
85     {
86        try {
87           String query = inputQuery.getText();
88
89           statement = connection.createStatement();
90           resultSet = statement.executeQuery( query );
91           displayResultSet( resultSet );
92        }
93        catch ( SQLException sqlex ) {
94           sqlex.printStackTrace();
95        }
96     }
97
98     private void displayResultSet( ResultSet rs )
99        throws SQLException
100    {
```

```
101          // position to first record
102          boolean moreRecords = rs.next();
103
104          // If there are no records, display a message
105          if ( ! moreRecords ) {
106             JOptionPane.showMessageDialog( this,
107                "ResultSet contained no records" );
108             setTitle( "No records to display" );
109             return;
110          }
111
112          Vector columnHeads = new Vector();
113          Vector rows = new Vector();
114
115          try {
116             // get column heads
117             ResultSetMetaData rsmd = rs.getMetaData();
118
119             for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
120                columnHeads.addElement( rsmd.getColumnName( i ) );
121
122             // get row data
123             do {
124                rows.addElement( getNextRow( rs, rsmd ) );
125             } while ( rs.next() );
126
127             // display table with ResultSet contents
128             table = new JTable( rows, columnHeads );
129             JScrollPane scroller = new JScrollPane( table );
130             Container c = getContentPane();
131             c.remove( 1 );
132             c.add( scroller, BorderLayout.CENTER );
133             c.validate();
134          }
135          catch ( SQLException sqlex ) {
136             sqlex.printStackTrace();
137          }
138       }
139
140       private Vector getNextRow( ResultSet rs,
141                               ResultSetMetaData rsmd )
142           throws SQLException
```

```
143        {
144          Vector currentRow = new Vector();
145
146          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
147            switch( rsmd.getColumnType( i ) ) {
148              case Types.VARCHAR:
149              case Types.LONGVARCHAR:
150                  currentRow.addElement( rs.getString( i ) );
151                break;
152              case Types.INTEGER:
153                  currentRow.addElement(
154                    new Long( rs.getLong( i ) ) );
155                break;
156              default:
157                System.out.println( "Type was: " +
158                  rsmd.getColumnTypeName( i ) );
159            }
160
161          return currentRow;
162        }
163
164        public void shutDown()
165        {
166          try {
167            connection.close();
168          }
169          catch ( SQLException sqlex ) {
170            System.err.println( "Unable to disconnect" );
171            sqlex.printStackTrace();
171          }
172        }
173
174        public static void main( String args[] )
175        {
176          final DisplayQueryResults app =
177            new DisplayQueryResults();
178
179          app.addWindowListener(
180            new WindowAdapter() {
181              public void windowClosing( WindowEvent e )
182              {
183                app.shutDown();
```

```
184                 System.exit( 0 );
185             }
186         }
187     );
188 }}
```

تمارين

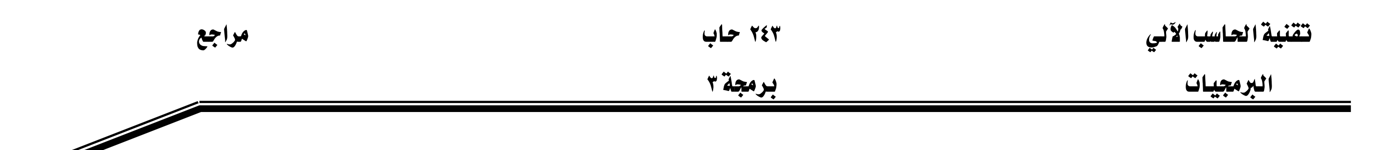
س١) أيها يوفر إمكانيات أكبر في معالجة البيانات: استخدام الملفات لتخزين البيانات أم الاتصال بقاعدة بيانات محددة، ولماذا؟

س٢) لقد رأيت كيف تعرف مشغل لربط قاعدة بيانات اكسس ببرنامج جافا. قم الآن بتعريف مشغل لربط قاعدة بيانات أوراكل ببرنامج جافا، وذلك باتباع نفس الخطوات الموضحة في الدرس مع اختيار قاعدة بيانات أوراكل بدلا من اكسس.

س٣)

١. عدل في المثال الثاني، المشروح في هذا الدرس بحيث يستخدم المشغل الذي قمت بتعريفه في السؤال الثاني.

٢. عدل واجهة التطبيق في نفس المثال لتتلائم مع الأعمدة المرتجعة من قاعدة البيانات أوراكل الخاصة بك.



المرجع

- Java How to Program, Deitel and Deitel, Fourth Edition

المحتويات

تقدر المؤسسة العامة للتعليم الفني والتدريب المهني الدعم

المالي المقدم من شركة بي آيه إي سيستمز (العمليات) المحدودة

GOTEVOT appreciates the financial support provided by BAE SYSTEMS

BAE SYSTEMS